2004

# AUTOMATED SYNTHESIS OF VIRTUALBLOCKS FOR INTERFACING SYSTEM UNDER TEST

Andrew Hai Liang She
*University of Kentucky*, ahshe0@uky.edu

ABSTRACT OF THESIS

AUTOMATED SYNTHESIS OF VIRTUALBLOCKS FOR
INTERFACING SYSTEM UNDER TEST

In this thesis, I/O signal recognizers, called VIRTUALBLOCKS, are synthesized to interface with a SYSTEM UNDER TEST (SUT). Methods for automated synthesis of virtualblocks allow us to simulate environment interfaces with SUT and also perform fault detection on SUT. Such methods must be able to recognize incoming sequences of signals from SUT, and upon the signal recognition determine the proper outgoing sequences of signals to SUT. We characterize our systems into four distinctive systems: system under test, AUXILIARY SYSTEM, controller and external environment. The auxiliary system is represented as a form of condition system Petri net (virtualblocks) and interacts with SUT along with the interaction among the controller and the external environment. Fault detection is performed by subsystems called DETECTBLOCKS synthesized from the virtualblocks. We present construction procedures for virtualblocks & detectblocks and discuss the notion of LEGALITY and DETECTABILITY. Finally, we illustrate our approach using a model of a scanner control unit.

KEYWORDS: Auxiliary System, Petri Nets, Fault Detection, Condition Systems,
System Under Test

Andrew Hai Liang She

12/01/04

AUTOMATED SYNTHESIS OF VIRTUALBLOCKS FOR
INTERFACING SYSTEM UNDER TEST

By
Andrew Hai Liang She

Dr. Larry E. Holloway
Director of Thesis.

Dr. Yu-Ming Zhang
Director of Graduate Studies.

12/01/04

RULES FOR THE USE OF THESES

Unpublished dissertations submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name Date

THESIS

Andrew Hai Liang She

The Graduate School
University of Kentucky
2004

# AUTOMATED SYNTHESIS OF VIRTUALBLOCKS FOR INTERFACING SYSTEM UNDER TEST

THESIS

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By
Andrew Hai Liang She

Lexington, Kentucky

Director: Dr. Lawrence E. Holloway,
Professor of Electrical & Computer Engineering,
University of Kentucky,

Lexington, Kentucky
2004

# ACKNOWLEDGMENTS

I thank the Graduate School and the Center for Robotics and Manufacturing Systems at the University of Kentucky for giving me a Kentucky Graduate Scholarship, a research assistantship and unlimited usage of computing, printing, and office facilities all through the process of my master degree studies.

My parents have been given me a lot of love and support throughout my studies in the USA. They have had to live with many years of separation from me while I have been involved in my academic pursuits in the USA.

Finally, I would like to express my sincerest thankfulness to my thesis advisor, Dr. Larry Holloway for giving me this opportunity to work with him. His extraordinary knowledge of the control world has inspired me to do this thesis. Because of his guidance and open mind, I have been able to make my dream come true of completing this master's thesis program.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF FILES

**AHLSthes.pdf** **671 KB**

# Chapter 1

# Introduction

In this thesis, we address issues on modeling and fault detection for a class of modeling systems called condition systems Petri nets. We first present the theoretical basis for condition systems. We then establish a concept of modeling for interfacing and define the notion of legality and detectability. The construction procedures for virtualblocks and detectblocks will also be presented in the thesis. We conclude this thesis by illustrating how we apply our modeling and fault detection methodologies into a scanner control unit application.

In the next section, we will present the background information and current issues on modeling and fault detection in the literature of system engineers, control system researchers, computer scientists and reliability engineers. We end this chapter with a discussion of our approach for the methodologies and an overview of the rest of the thesis.

## 1.1 Background

According to system and control theory, there are two prominent features in the very definition of SYSTEM. First, a system consist of interacting components/subsystems, and second a system is associated with a functionality. As system engineers, we are interested in the quantitative analysis of systems and therefore we seek a mathematical model of an actual physical system. There are two distinct physical phenomena to be modeled: physical systems which are modeled by mathematical equations, and physical signals which are modeled by mathematical functions [PP99].

Systems are further categorized into different classes based on their own unique characteristics. In system classifications, systems can generally be classified into static systems and dynamic systems. A static system is a system where output is in-

dependent of past values of input, and whereas a dynamic system is a system where output determination generally requires "memory" of input history. From dynamic systems, systems can be further classified depending on their stationarity (time-varying, time-invariant), linearity (linear, nonlinear), state space (continuous, discrete), state transition mechanism (time-driven, event-driven), predictability (deterministic, stochastic) and time sample path (continuous, discrete). A detailed description of those system classifications can be found in [G85].

Historically, scientists and engineers have concentrated on studying natural phenomena which are well modeled by laws of physics, chemistry, astronomy and other physical sciences. So we typically encounter with quantities such as velocity and acceleration of a solid particle, temperature rates of fluids and gases, gravity force of a planet and etc. All of these quantities are considered "continuous variables" because the state space of these variables are both continuous and comprised of real number. Based on these system characteristics, mathematical techniques such as calculus had been developed to perform system modeling. To use these mathematical models, there are two key properties that systems must satisfy: state space is a continuum and the state transition mechanism is time-driven. This class of systems is also referred as CONTINUOUS-VARIABLE DYNAMIC SYSTEMS (CVDS).

But nowadays, we encounter systems that are inefficient to be modeled mathematically by continuous variables. First of all, state space for such systems are "discrete", typically involving integer numbers. And second, their state transition mechanism depends on instantaneous "events". Such systems include computer systems, communication systems and manufacturing systems. Based on this fact, a class of dynamic systems: Discrete Event Dynamic Systems, or just DISCRETE EVENT SYSTEMS (DES) is being introduced. Discrete event systems are systems whose state space are discrete and state changes can only occur as a result of asynchronously occurring instantaneous events over time. There are basically three levels of abstraction in DES: Logical (untimed), Timed and Timed Stochastic. The choice of the appropriate level of abstraction depends on the objective of the analysis.

As for modeling formalism in discrete event systems, two major formalisms are AUTOMATA and PETRI NET [KG95],[MA98],[ZV99],[ZD93],[E03]. Each of these formalisms have their own unique properties and advantageous over the issues of concurrency, modularity, state explosion and decidability. A detailed comparison of Petri nets and automata can be found in [CL99].

Detection of system failure plays a crucial role in protecting human life and improving the overall performance of industrial processes through reducing the risk of product failure and time to market pressure. Fault Diagnosis has been the subject of extensive research among various research communities due to the fact that the swift evolution of computing, communication and industrial technologies in the era of information revolution has brought the proliferation of new dynamic systems which is often highly complex and gigantic. Thus the increasing complexity in technological systems have necessitated the development of systematic methods for accurate and reliable fault detection system.

First of all, we will review the concept of "FAULT". By reading through the literature in the field of fault diagnosis, one can easily discover that the terminology of fault in this field is not consistent. In [HA02], fault is the inconsistency of system observation with the expected modeled behaviors. Fault on the other hand is considered to be an unobservable event in [SSL96]. Fault in [HCJK03] represents a normal occurrence or an inherent characteristic of system which is inevitable in the existing industrial environment. According to SAFEPROCESS Technical committee of control engineering society, there are distinctions among the very definition of fault, failure, malfunction, error, disturbance and etc. For example, fault is defined as an unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable, usual or standard condition. Failure, "a permanent interruption of a system's ability to perform a required function under specified operating conditions". Error, "a deviation between measured or computed value of an output variable and its true or theoretically correct one". Symptom, "a change of an observable quantity from normal behavior". More definitions and terminologies can be found in nomenclature section of [SFP03].

Despite the deviation in the terminology of fault among different researchers, the term "FAULT DIAGNOSIS" is also being treated differently. While in [SSL95] the authors define failure diagnosis as the detection of failure events and identification of the type of failure events through the record of observable events, in [J04] the author distinguishes the definition of fault diagnosis with different meanings by defining fault detection as the determination that the system behavior is different from allowed behavior and fault diagnosis as localizing or identifying the fault.

The issues of fault diagnosis are well explored problems in reliability engineering, computer science and control system research, in particularly discrete event

system. Fault diagnosis using fault tree analysis has been studied in detail by reliability engineers. The analysis starts by considering an overall failure event and working down the tree to identify the roots/parts of failure [O'C81]. Expert systems and model based reasoning schemes for diagnosis have been proposed by computer scientists. Expert systems are generally being applied in the case when it is difficult to design and obtain a model for a particular system.

In addition to MODEL-FREE methods of expert systems from computer scientists, quantitative-analytical model based methods have been extensively used by control system researchers. In MODEL-BASED fault detection, a traditional approach to fault detection is based on hardware or physical redundancy methods which require the use of multiple sensors and actuators to measure and control a particular variable. A typical voting technique is then applied to the hardware redundant system to decide whether a system fault has occurred. One of the main problems of the traditional hardware redundancy methods is the extra cost incurred from the use of multiple redundant hardware in the system. Due to the conflict between adding extra cost and reliability, the analytical or functional redundancy methods have gradually replaced this traditional approach.

In the analytical redundancy scheme, a mathematical model system will be obtained from an actual physical system/plant. Input variables will then be applied into the system models and actual system. The output variables from these systems will be gathered and compared. Ideally the system behavior of model system should mimic the actual system. The difference generated from the comparison of variables will be called a symptom or residual signal. If the system is operating normally then the residual signal should be zero. Thus the residual signal is used to determine whether a fault has occurred or not. These model-based diagnosis schemes rely on continuous-variable models such as differential and difference equations. Examples of the methodologies include observer-based approaches, parameter estimation and parity vector methods. The observer-based methods work by generating residuals for output variables with fixed parametric models. Fixed parametric or non parametric models are used under parity equations method and adaptive nonparametric or parametric models are used under parameter estimation methodology [SFP03].

In [ZKW03], the authors state that for the purpose of only detecting and diagnosing some particular unique failure, detailed continuous-variable models as

in analytical redundancy schemes are often unnecessary. Under these conditions, discrete event system models are usually sufficient as system models in terms of information integrity and usually provide a more convenient way to model due to the nature of discrete systems comparing to continuous systems. DES techniques in fault detection generally require the use of models to model faulty behaviors and then use a form of detector system to determine a proper detection from a given set of observed events.

## 1.2 Approach

In this thesis, there are two salient issues to be tackled: modeling & fault detection of system. On the modeling issue, the goal of our system modeling is to simulate the auxiliary system which interfaces with SUT. We begin our modeling approach by first analyzing the details and properties of our given systems which mainly consist of system under test, auxiliary system and the controller. Next from our analysis, we will then determine the appropriate modeling formalism and methodology for our given systems. In terms of modeling formalism, we will use condition system Petri net models as our systems modeling formalism. Condition systems are a form of Petri net where systems are composed of subsystems which interact through condition signals. The advantages of using condition system Petri net will be discussed in the following chapter. The auxiliary system modeling methodologies are relied on the automated synthesis of virtualblocks which capable of recognizing input signals from SUT and providing appropriate output signals to SUT. Virtualblocks under our modeling methodology are designed to recognize and output two different type of signals from SUT which are either clocked signals or non-clocked signals.

As for fault detection, we will specifically focus on model-based OFFLINE PASSIVE fault detection only. This means that our methodologies involve modeling of systems and fault detection systems that are derived from auxiliary system before the given systems are on the line of work. Our methodologies also do not use any test inputs to detect system failure. In our approach, a fault, failure or any other faulty terms will be treated the same and the term is defined as an inconsistency between observed system behavior from system under test with the expected system behavior of auxiliary system. Fault detection will be defined as the determination that the system under test is not behaving as expected according to the model of

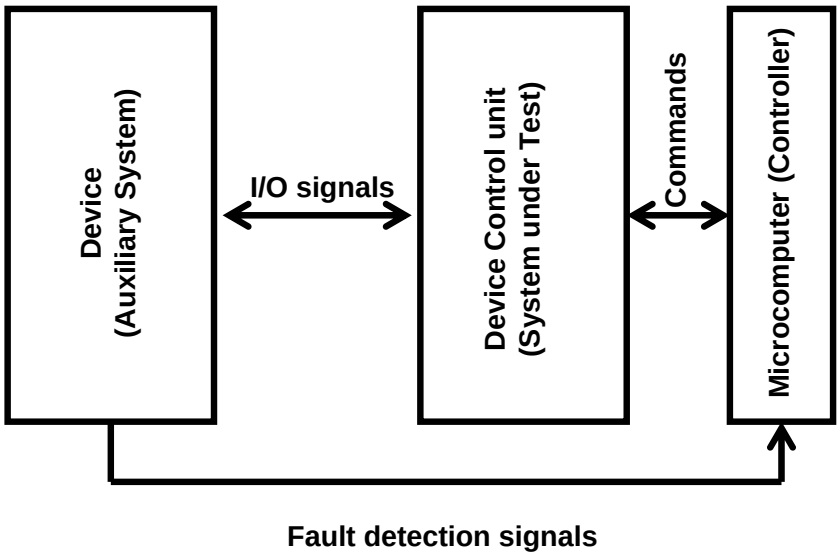


Figure 1.1: Systems within the Methodology

the auxiliary system. In addition, our approaches vary in the respect that we do not require the modeling of fault and thus drastically simplify the modeling process. Our fault detection methodologies rely on detectblocks which are synthesized from virtualblocks. Detectblocks are responsible for detecting whether a given sequences of signals are recognizable by virtualblocks. If virtualblocks are unable to recognize the signals then detectblocks will issue a fault detection signal.

In general, systems considered within the thesis consist of a physical device, device's control unit and a microcomputer. The physical device interacts with its control unit through a common signal interface. The task of the microcomputer is to control both the device and device's control unit and determine their correct system operations. Such physical system design testing has several disadvantages in term of flexibility and observability. It is impractical to use a real system in design testing due to the time, cost and future design constraint impose on it. From these disadvantages, we are motivated to develop a systematic methodology which utilizes software model in system design testing. The main goal of the methodology is to create a system model out of the physical device. The system model of the device will not only provide a virtual system to the device's control unit but also perform fault detection on the control unit. Initially we intend to design a complete model from the physical device. During the modeling process, we encounter several

modeling issues such as insufficient system information and the existence of CVDS environment within the system components where we are unable to obtain a complete model from the device. Due to these issues, we decided to create a model to simulate the interface between device's control unit and physical device instead of a complete model. To achieve these goals, there are two main tasks for the device model: 1. Responds to control unit's excitations with appropriate responses and 2. Detects fault among control unit's excitations. The device model will be modeled and simulated in a modular approach. Such modular approach allows the system to be modeled within the Spectool (a type of control synthesis software tool) framework which will implement automated synthesis of software code for the device model. In addition, it will also permit the performance of formal model analysis in future research. Under our methodology framework, device's control unit will be denoted as system under test (SUT) and the device will be denoted as auxiliary system. Microcomputer will be considered as a controller.

The thesis is organized as follows. In chapter 2, we present the background information of condition system languages & model and condition subsystem models composition & modularity. We will define the concept of modeling for interfacing in chapter 3, and in chapter 4 we present the notion of legality, virtualblocks and also construction procedures for virtualblocks. In chapter 5 the notion of detectability, detectblocks and construction procedures for detectblocks are illustrated. Applications for our modeling and fault detection methodologies will be illustrated in chapter 6. Finally, chapter 7 will be our thesis conclusion.

# Chapter 2

# Condition Systems

We present our approaches for modeling & fault detection method which rely on a form of modeling formalism namely, CONDITION SYSTEM. Condition system is a form of Petri net modeling formalism with explicit inputs and outputs called CONDITION SIGNALS. These explicit I/O features of the system allow us to represent a system as a collection of subsystems which interact through condition signals. The condition system framework is a subset of the condition/event models developed by Sreenivas and Krogh where there are two classes of input output signals for a C/E system: condition signals and event signals [SK91].

One of the main advantages of using condition system Petri nets is the ability to avoid state space problem in modeling of large and complex systems. With event communication(automata), the traditional DES approach, modeling of a huge sophisticated system requires synchronous composition of subsystem models which will lead to state explosion. Modeling formalisms emphasizing state communication such as condition systems can easily overcome this issue. The well defined notions of input and output in condition system framework consequently allow the system model to exhibit clear cause & effect relationships. The dynamics within the subsystems can be defined independent of each others due to these well defined interfaces, and this would simplify the system model construction by allowing the reuse of subsystem models.

This chapter is presented for the purpose of providing background knowledge on condition systems which is required for the understanding of later chapters. The chapter is organized into four sections. In the first section of the chapter, we will define the condition system languages. We then present the model of condition system in section 2.2. In section 2.3, we discuss about the composition of condition system models. In the last section, we present the modularity of condition system models by introducing a special kind of condition system, Specification Block.

## 2.1 Condition System Languages

In this section, we introduce notions and notations of the languages generated by condition systems found in [HA98a], [HA98b], [HGSA00] & [HA02]. The systems that we consider interact with each other and their external environment through conditions. A condition is a signal that either has value "TRUE", or "FALSE". A condition with a "true" value would mean that the particular condition is valid and vice versa for the condition with "false" value. Let *AllC* be the universe of all conditions, such that for each condition c in *AllC* , there also exists a negated condition denoted $\neg c$, where $\neg(\neg c) = c$. Such notation of negation will contribute to a form of condition signal property: CONTRADICT; where a condition signal c is said to be contradict to condition signal $\neg c$. We will define $\mathcal{T}rueC$ as set of conditions (C) where their condition values are "true". Therefore $\neg\mathcal{T}rueC$ is defined as set of conditions (C) at a given time where their condition values are "false". Also note that condition $\emptyset$ will be defined as condition set that does not have any conditions.

Next we will introduce another condition signal property that first appeared in [GH00]. A condition signal c is said to be EXCLUSIVE to another condition signal c', if at most one of c, c' can be true at any time. Note that c' is not necessarily the same as $\neg c$. Two condition sets are exclusive if each set contains at least one condition signal exclusive to another condition signal in the other condition set. We assume that there exists a designation of exclusive condition signals over *AllC* . Such condition signal property is essential for avoiding system conflict among condition system model which will be illustrated in the following section. Note that any contradicting signals are necessarily exclusive.

System behavior of a particular condition system can be described by sequence of condition sets. A condition set sequence, called a C-SEQUENCE, is a finite length sequence of condition sets. A C-sequence from a typical condition system will indicate a string of ordered condition sets which is valid over certain period of time and thus describing the system behavior of condition system during that time frame. Each condition set sequence is of the form $(C_0C_1...C_n)$ for some integer n and sets $C_i \subseteq$ *AllC* for all $0 \leqslant i \leqslant n$. From the concept of condition signal property: Contradict, a C-sequence is said to be CONTRADICTION FREE if for each $C_i$ for any $c \in C_i$, then $\neg c \notin C_i$. Given two C-sequences $s_1$ and $s_2$, the expression $s_1s_2$ will indicate the concatenation of $s_2$ on the end of $s_1$, and this will allow the formation of C-sequences set. A set of C-sequences is called a language, and the set consisting of

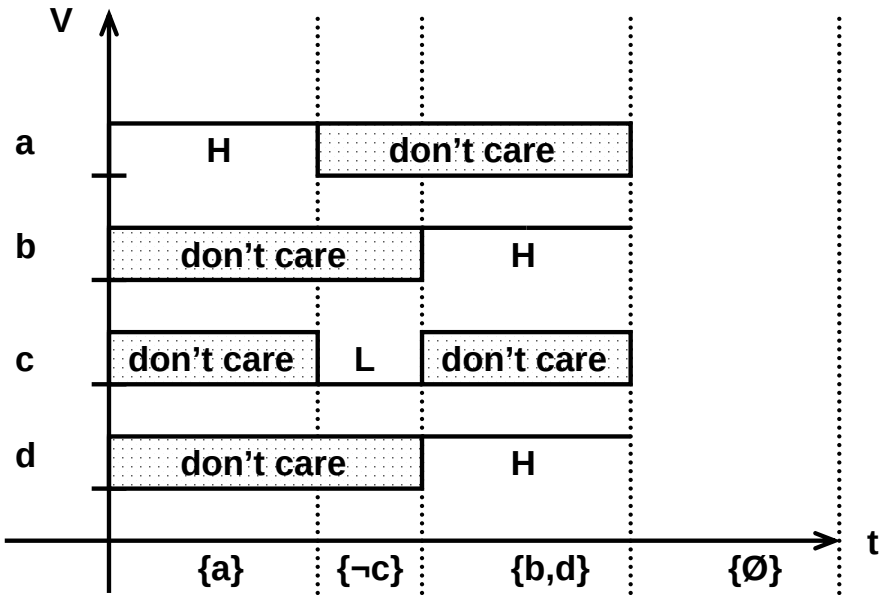


Figure 2.1: Examples of voltage signal time lines corresponding to the C-sequence $s = (\{a\}\{\neg c\}\{bd\}\{\emptyset\})$

all C-sequences is denoted $\mathcal{L}$.

A C-sequence can be viewed as a sequence of conditions that must be true over certain specified though ordered time periods. Given a C-sequence $s = (C_0C_1...C_n)$ and some $0 \leq i \leq n$, $C_i$ represents a subset of conditions (or negated conditions) that are true for some (possibly non-unique) period of time. $C_i$ does not have to include all true conditions over the time period. However, the time period that $C_i$ represents must follow immediately after the time period represented by $C_{i-1}$, and must be followed immediately by the time period represented by $C_{i+1}$. This is further illustrated in the high level voltage signal modeling of Figure 2.1. Note that the condition $a$ might be true throughout the time line, but does not have to be listed in all condition sets in the sequence. This is analogous to a "do not care" condition on its value when it is not specified.

### 2.1.1 Descriptive Ordering

Next, we will introduce the notion of DESCRIPTIVE ORDERING from [HA98b] which will allow us to compare elements of condition languages (sequences of condition sets). Elements of these languages are sequences of condition sets that are responsible for representing the evolution of a condition system. These condition sequences can also be used to specify the desired system behavior of a typical condition system. Each elements of the language will contain condition information of a particular condition system, and the ordering will be used to compare the richness of such information among each elements.

For the goal of simplicity in condition language analysis, we need to describe important characteristics of condition sequence without listing all the details of all condition activity within the C-sequence. A convenient way to characterize a C-sequence is through a partial ordering "$\leq$" which we had previously referred to as the descriptive ordering. Such ordering can be used to analyze and compare features of different C-sequences.

Definition 2.1 will formally define the notion.

**Definition 2.1** Define the DESCRIPTIVE ORDERING $\leq$ over condition sequences such that:

1. $(C_1 C_1') \leq (C_2)$ if $C_1 \subseteq C_2$ and $C_1' \subseteq C_2$.
2. $(C_1) \leq (C_2 C_2')$ if $C_1 \subseteq C_2$ and $C_1 \subseteq C_2'$.
3. Given C-sequences $s_1$, $s_1'$, $s_2$, and $s_2'$ such that $s_1 \leq s_1'$ and $s_2 \leq s_2'$, then $s_1 s_2 \leq s_1' s_2'$.
4. If $s_1 \leq s_2$ and $s_2 \leq s_3$, then $s_1 \leq s_3$.

From the definition above, we see that given sequences $s_1$ and $s_2$, if $s_1 \leq s_2$, then $s_1$ contains no more condition information in it than $s_2$, and $s_2$ can be said to be AT LEAST AS DESCRIPTIVE as $s_1$. If $s_1 \leq s_2$ and $s_2 \leq s_1$, then the sequences are said to be EQUIVALENT under the ordering, written as $s_1 \equiv s_2$. Statement 1 and 2 in the definition above establish the ordering based on subsets of condition sets. Statement 3 considers the concatenation of smaller ordered C-sequences, and statement 4 defines the ordering to be transitive. Conditions that are not listed are

considered "don't care" conditions. The descriptive ordering lets us omit consideration of specific conditions during periods when they are not of interest, while still allowing comparison of some basic sequencing characteristics.

**Example 2.1** To illustrate the descriptive ordering, consider a power control unit for a document scanning system with condition signals SCANNER ON, MOTOR ON and signals MU and MD to indicate that the scanner motor is moving in upward or downward position. Example C-sequences are as follows.

$s_1 = (\{\emptyset\}\{motor\ on, MD\})$
$s_2 = (\{motor\ on\}\{MD\}\{motor\ on, MD\})$
$s_3 = (\{motor\ on\}\{motor\ on\}\{MD\}\{motor\ on, MD\})$
$s_4 = (\{motor\ on, MU\}\{motor\ on, MD\})$
$s_5 = (\{motor\ on, MD\})$
$s_6 = (\{motor\ on, MU, scanner\ on\}\{motor\ on, MD, scanner\ on\}$
$\{motor\ on, MU, \neg scanner\ on\})$

The following relationships are true.

$$s_1 \leq s_2 \equiv s_3 \leq s_4 \leq s_6$$

$$s_1 \leq s_2 \equiv s_3 \leq s_5$$

Note that $s_5$ and $s_6$ are not comparable under the descriptive ordering since $\{motor\ on, MD\} \not\subseteq \{motor\ on, MU, scanner\ on\}$ and vice versa.

Let (*AllC*) be the C-sequence of length one that consists of all conditions (including negations). Note that it is inherently contradictory. Let $(\{\emptyset\})$ be the C-sequence of length one that consists of no conditions. The following results can be shown.

**Lemma 2.1 (HA98b)** The following statements are true:

1. $s \leq$ (*AllC*) for any C-sequence $s$.
2. $(\{\emptyset\}) \leq s$ for any C-sequence $s$.
3. $(C) \equiv (CC) \equiv (CCC) \equiv (C^n)$ for any condition set C and any $n > 0$.

4. Given C-sequences $s_1$ and $s_2$ and condition set C, $s_1 C s_2 \equiv s_1 C C s_2$.

5. Given C-sequences $s_1$ and $s_2$ and condition set C, $C'$, if $C \subseteq C'$, then $s_1 C s_2 \leq s_1 C' s_2$.

Note that statement 1 of lemma 2.1 says that the condition sequence consisting of the set of all conditions (and their negations) being true is the most descriptive of all C-sequences (but it is contradictory). Statement 2 says that the C-sequence consisting of just an empty set of conditions is the least descriptive C-sequence, since it says nothing about the truth value of any condition at any time. Statement 3 says that any finite nonzero length sequence is equivalent to any other finite nonzero length sequence of the same condition set. Statement 4 says that duplication of a condition set within a sequence results in an equivalent sequence. Statement 5 considers two sequences that differ only in a single condition set, where the set in the first sequence is a subset of the set in the second sequence. The statement then says that the second sequence is at least as descriptive.

### 2.1.2 Observability

Finally we conclude this section with a brief definition of OBSERVABILITY over conditions which was initially introduced in [HA98a]. Let $C_{obs} \subseteq AllC$ be a set of conditions which can be observed, where it is implied that if $c \in C_{obs}$ then $\neg c \in C_{obs}$. For any $c \in C_{obs}(\mathcal{G})$, this will implied that the condition signal c is observable with respect to $\mathcal{G}$. These observed conditions can either be the inputs to $\mathcal{G}$ or outputs of $\mathcal{G}$. Note that the internal state of $\mathcal{G}$ is not always observable, and their observability will depend on the synthesis of state observer within the system [GH00],[GH01].

Next we will associate observed condition set, $C_{obs}$ with C-sequence, s in the following definition.

**Definition 2.2 OBSERVABILITY :** Given a C-sequence $s = (C_0 C_1 ... C_n)$ for some integer n and some condition set $C_{obs} \subseteq AllC$, define the projection of s onto C, denoted $s|_C$ as $s|_{C_{obs}} = ((C_0 \cap C_{obs})(C_1 \cap C_{obs})...(C_n \cap C_{obs}))$.

Therefore $s|_{C_{obs}}$ is also known as the observed system behavior of a particular condition system.

From the definition 2.2 we get the following basic result presented in lemma 2.2:

**Lemma 2.2 (HA02)** For any $s \in \mathcal{L}$ and any $C \subseteq \mathit{AllC}$,

$s\,|_{C_{obs}} \leq s$.

C-sequence $s$ is said to be at least as descriptive to observed C-sequence $s\,|_{C_{obs}}$. Note that, if $s\,|_{C_{obs}} \equiv s$, these would imply that all the condition sets in C-sequence $s$ are observable.

## 2.2 Condition System Model

Condition system $\mathcal{G}$ is defined as a form of Petri net that requires conditions for enabling of transitions, $\mathcal{T}_\mathcal{G}$ and outputs condition signals through places, $\mathcal{P}_\mathcal{G}$ according to its markings $m$. Definition 2.1 from [HGSA00] formally defines condition systems that we consider for this thesis.

**Definition 2.3** A condition system $\mathcal{G}$ is characterized by a set of states $M_\mathcal{G}$, a next state mapping $f_\mathcal{G} : M_\mathcal{G} \times 2^{\mathit{AllC}} \longrightarrow 2^{M_\mathcal{G}}$, and a condition output mapping $g_\mathcal{G} : M_\mathcal{G} \longrightarrow 2^{\mathit{AllC}}$. In this paper, we assume that $M_\mathcal{G}$, $f_\mathcal{G}$, and $g_\mathcal{G}$ are defined through a form of Petri net consisting of a set of places $P_\mathcal{G}$, a set of transitions $T_\mathcal{G}$, a set of directed arcs $A_\mathcal{G}$ between places and transitions, and a condition mapping function $\Phi_\mathcal{G}(\cdot)$, where $(\forall p)\Phi_\mathcal{G}(p) \subseteq \mathit{AllC}$ maps output conditions to each place, and $(\forall t)\Phi_\mathcal{G}(t) \subseteq \mathit{AllC}$ maps ENABLING CONDITIONS to each transition. The net is related to $M_\mathcal{G}$, $f_\mathcal{G}$ and $g_\mathcal{G}$ in the following manner:

1. THE STATES ARE THE MARKINGS OF THE PETRI NET: each state $m \in M_\mathcal{G}$ is a function over $\mathcal{P}_\mathcal{G}$ that represents a mapping of nonnegative integers to places.

2. THE OUTPUT CONDITIONS RESULT FROM MARKED PLACES: for any $m \in M_\mathcal{G}$, $g_\mathcal{G}(m) = \{c \mid \exists p \text{ s.t. } c \in \Phi_\mathcal{G}(p) \text{ and } m(p) \geq 1\}$

3. NEXT-STATE DYNAMICS DEPEND ON STATE ENABLING AND CONDITION ENABLING: for any $m \in M_\mathcal{G}$ and any set of conditions $\mathit{TrueC} \subseteq \mathit{AllC}$, $m' \in f_\mathcal{G}(m, \mathit{TrueC})$ if and only if there exists some transition set $T$ such that

   (a) $T$ is STATE-ENABLED, meaning $(\forall p \in P_\mathcal{G})\ m(p) \geq |\{t \in T | p \text{ is input to } t\}|$

   (b) $T$ is CONDITION-ENABLED, meaning $(\forall t \in T)\ \Phi_\mathcal{G}(t) \subseteq \mathit{TrueC}$

   (c) the next marking $m'$ satisfies $\forall p \in P_\mathcal{G}$,
   $m'(p) = m(p) - |\{t \in T |\ p \text{ is input to } t\}| + |\{t \in T |\ p \text{ is output of } t\}|$

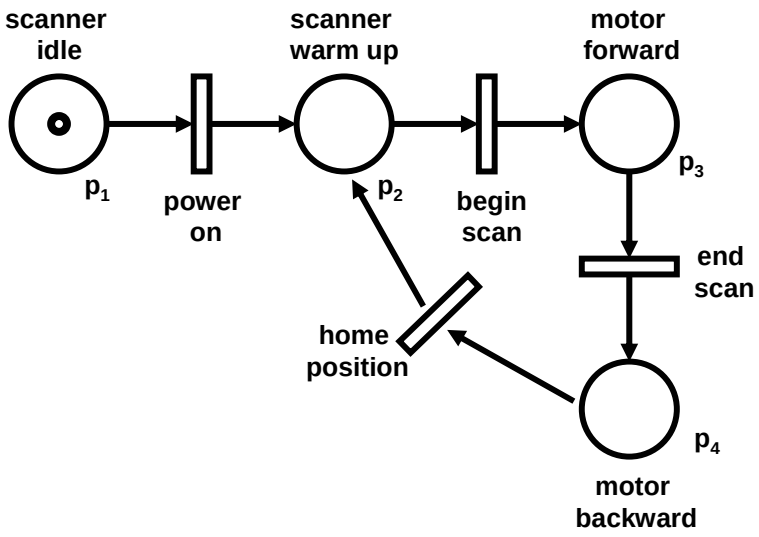


Figure 2.2: Example of condition system model for a scanner motor power control unit.

4. $M_{\mathcal{G}}$ IS CLOSED UNDER $f_{\mathcal{G}}(\cdot)$: if $m \in M_{\mathcal{G}}$ and $m' \in f_{\mathcal{G}}(m, TrueC)$ for some $TrueC \subseteq AllC$, then $m' \in M_{\mathcal{G}}$.

In statement 2, we assume all conditions that are output from G will have value defined by the marking of $\mathcal{G}$. Thus, if a condition c is forced true on one marking, then it will also be forced to either true or false for all other markings either through the function g or defaulting to a known value. We note that items in 3a and 3c above correspond to standard Petri net state enabling and firing of a transition set, respectively. Item 3b adds an additional transition set enabling constraint that the input conditions to each transition must also be within the considered set $TrueC$ of true conditions.

We will define enabling conditions of transitions as the input condition set for a condition system $\mathcal{G}$, $C_{in}(\mathcal{G})=\{c \in \Phi_{\mathcal{G}}(t) \mid t \in \mathcal{T}_{\mathcal{G}}\}$. Similarly we define the conditions of a place as the output condition set for the condition system $\mathcal{G}$, $C_{out}(\mathcal{G})=\{c \in \Phi_{\mathcal{G}}(p) \mid p \in \mathcal{P}_{\mathcal{G}}\}$.

**Example 2.2** Consider the condition system model shown in Figure 2.2. The net shown represents a simple scanner mechanism under its motor control unit. There are three basic operational buttons on the scanner: "power on", "begin scan" & "end

scan" and they are controlled by the user. The scanner motor control unit can either be in the state of "scanner idle", "scanner warm up", "motor forward" or "motor backward". The conditions associated with the transitions represent input conditions to the system. The conditions on the places are output conditions from the system. If a place is marked then its associated output condition is "true". According to Definition 2.3, the marking shown in the figure is $m_0$ = [1 0 0 0] and $g_{\mathcal{G}}(m_0)$ = { scanner idle }. Under the input condition set $C_{in}(\mathcal{G})$ = { power on }, the transition from $p_1$ to $p_2$ is both state enabled and condition enabled, so $f_{\mathcal{G}}(m_0, C)$ = { [0 0 0 1], [0 0 1 0] }. Note that when the scanner is in the state of "motor backward", an internal command "home position" will be issued within the scanner control unit to bring the scanner back to the "scanner warm up" state.

The next lemma follows directly from the Definition 2.3.

**Lemma 2.3 (HGSA00)** Consider a condition system $\mathcal{G}$, with marking $m$ and next state mapping $f_{\mathcal{G}}$. The following statements are true:

1. Given condition sets C and C′, if $C \subseteq C'$, then $f_{\mathcal{G}}(m, C) \subseteq f_G(m, C')$;
2. Given condition sets C and C′, if $C \cap C_{in}(\mathcal{G}) = C' \cap C_{in}(\mathcal{G})$, then $f_{\mathcal{G}}(m, C) = f_{\mathcal{G}}(m, C')$;
3. For any true condition set TrueC, $m \in f_{\mathcal{G}}(m, \mathrm{TrueC})$.

The first statement relates the next state marking with subsets of condition sets and the second statement explores on relationship of input condition sets with the next state marking. The third statement of the lemma 2.3 is true because the set of transitions T in Definition 2.3 can be an empty set which means that the next state dynamics is independent of state enabling and condition enabling of the corresponding set of transitions, T due to the issue of timing delay in real world modeling.

**Definition 2.4** Given a condition system $\mathcal{G}$ and a marking $m_0$, define the language $L(\mathcal{G}, m_0) \subseteq \mathcal{L}$ to be the set of condition set sequences such that $(C_0C_1C_2...C_n) \in L(\mathcal{G}, m_0)$ if there exists some marking sequence $(m_0m_1...m_k)$ and index mapping function $j(i)$ with $j(0) = 0$, $j(k) = n$ such that:

1. MARKINGS EVOLVE ACCORDING TO CONDITIONS:

$m_{i+1} \in f_{\mathcal{G}}(m_i, C_{j(i)})$ for $0 \leq i \leq k-1$.

2. OUTPUT CONDITIONS RESULT ONLY FROM THE MARKING:

   $g_{\mathcal{G}}(m_i) = C_{j(i)} \bigcap C_{out}(\mathcal{G})$.

3. SEQUENCING IS MAINTAINED:

   A marking $m_{i+1}$ either maps to condition sequence element $C_{j(i)}$ corresponding to prior marking $m_i$ in the marking sequence, or it maps to the next condition sequence element. More formally, for any $0 \leqslant i < k$, $j(i+1) = j(i)$ or $j(i+1) = j(i) + 1$.

The above definition deserves some explanation. The notation $C_{j(i)}$ indicates the condition set associated with the $i$th marking. From statement 1, marking $m_i$ will evolve to marking $m_{i+1}$ only if it is enabled under condition set $C_{j(i)}$. Statement 2 states that the output conditions in set $C_{j(i)}$ correspond to the marking $m_i$. For statement 3, there are only two possibilities for condition set associates with marking $m_{i+1}$:it can either be the condition set that was associated with the previous marking $m_i$ or the next condition set following immediately from the condition set of previous marking. In this way, the condition sequencing is maintained.

The marking sequence and condition set sequences have different indices because the mapping between the sequence is not necessarily one-to-one. A marking could change from $m_i$ to $m_{i+1}$, but $g(m_i)$ and $g(m_{i+1})$ could be the same. Thus, it is possible that both markings could correspond to the same condition sets in the C-sequence. This then implies that there could be fewer condition sets in the C-sequence than distinct markings in the corresponding marking sequence.

On the other hand, note that for any $m$ and any C, $m \in f_{\mathcal{G}}(m, C)$, which implies that there is no transition firing. From statement 3, then it is possible that $m_{i+1} = m_i$. Under these circumstances, there will be more condition sets in the C-sequence than distinct markings in the corresponding marking sequence, Finally, we point out that $L(\mathcal{G}, m_0)$ is obviously prefix-closed (excluding the empty prefix).

From the definition 2.4 we get the following basic result:

**Lemma 2.4 (HA98a)** For any $\mathcal{G}$ with any marking $m$ and for any two C-sequences $s_1$, $s_2$ such that $s_1 \in L(\mathcal{G}, m)$, the following are true:

1. if $s_2$ is a prefix of $s_1$, then $s_2 \in L(\mathcal{G}, m)$

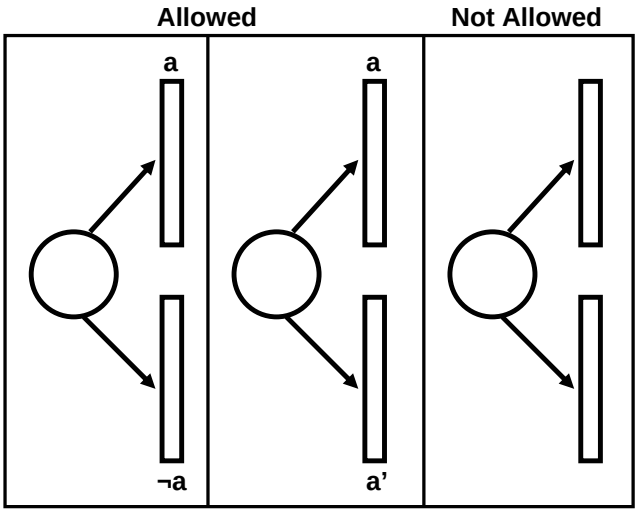


Figure 2.3: A simple chart showing some of the structural configurations which are allowed and not allowed for condition systems satisfying property deterministic

2. if $s_1 \leq s_2$ and $s_1 |_{C_{out,\mathcal{G}}} \equiv s_2 |_{C_{out,\mathcal{G}}}$, then $s_2 \in L(\mathcal{G}, m)$.

3. if $s_2 \equiv s_1$, then $s_2 \in L(\mathcal{G}, m)$.

Note from the lemma 2.4, statement 1 states that the set $L(\mathcal{G}, m)$ is prefix closed. Statement 2 says a string that is more descriptive with the equivalent output conditions is also in the language, $L(\mathcal{G}, m)$.

In this thesis, our condition system model is subjected to the limitation related to one of the subclasses of Petri Net, Free-Choice Petri Nets [P81]. The class of systems we consider is best illustrated with the condition system property DETERMINISTIC.

The system property DETERMINISTIC will be formally defined in Definition 2.5:

**Definition 2.5 Property DETERMINISTIC:** A condition system satisfies Property DETERMINISTIC if the following is true:

1. Given a place p and the set $p^{(t)}=\{t \in T \mid t \text{ is output of } p\}$, if the set $p^{(t)}$ has more than one element, then for each $t, t' \in p^{(t)}$, there exist $c \in \Phi_G(t)$ and $c' \in \Phi_G(t')$ with c and c' exclusive with each other.

According to this limitation, a place cannot be an input to several transitions with one exception that when enabling condition of each transitions are exclusive

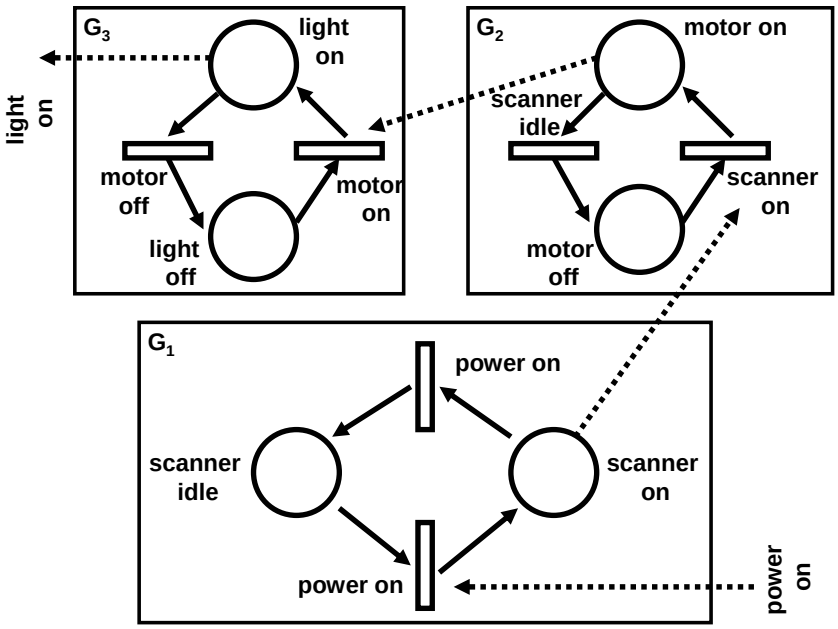


Figure 2.4: Condition subsystems model for the scanner power, lamp & motor control unit

or contradict among each other. But we do allow more than one place to be an input to a single transition. A place is also allowed to be an output from several transitions.

## 2.3 Composition of Condition System Models

A condition system can be subdivided into subsystems, where each subsystem is a condition system over a set of connected places and transitions which are disconnected from all other places and transitions. For the remainder of this thesis, we use the notation $\mathcal{G}$ to indicate the complete system, and the notation $\{G_1, ...G_n\}$ to indicate the set of subsystems in $\mathcal{G}$. Given an initial marking $m_0$ of $\mathcal{G}$, we let $m_{0,i}$ denote the marking over just the places in $G_i \in \mathcal{G}$.

Condition systems can also be composed to create other condition systems. Concurrent composition of condition systems is formally defined in the following definition:

**Definition 2.6 CONCURRENT COMPOSITION:** Given two distinct systems $G_1$ and

$G_2$ with markings $m_1$ and $m_2$, let $\mathcal{G} = G_1 \cup G_2$ the concurrent composition of $G_1$ and $G_2$ correspond to the simple unions of the systems, such that:

1. $\mathcal{G}$: $\mathcal{P}_{\mathcal{G}} = \mathcal{P}_{G_1} \cup \mathcal{P}_{G_2}$, $\mathcal{T}_{\mathcal{G}} = \mathcal{T}_{G_1} \cup \mathcal{T}_{G_2}$, $\mathcal{A}_{\mathcal{G}} = \mathcal{A}_{G_1} \cup \mathcal{A}_{G_2}$.
2. $G_1$: $\Phi_{\mathcal{G}}(x) = \Phi_{G_1}(x)$ for $x \in \mathcal{P}_{G_1} \cup \mathcal{T}_{G_1}$
3. $G_2$: $\Phi_{\mathcal{G}}(x) = \Phi_{G_2}(x)$ for $x \in \mathcal{P}_{G_2} \cup \mathcal{T}_{G_2}$

We assume $x$ cannot be in both $G_1$ and $G_2$. We will define the expression $m = m_1 \cup m_2$ such that $m(p) = m_1(p)$ for $p \in \mathcal{P}_{G_1}$ and $m(p) = m_2(p)$ for $p \in \mathcal{P}_{G_2}$.

The properties in the following lemma then result.

**Lemma 2.5 (HGSA00)** Given systems $G_1$ and $G_2$ with markings $m_1$ and $m_2$, the following properties are true:

1. Given some condition set $C$, some $m_1, m_1' \in M_{G_1}$, and some $m_2, m_2' \in M_{G_2}$, $m_1' \cup m_2' \in f_{G_1 \cup G_2}(m_1 \cup m_2, \mathrm{True}C)$ if and only if $m_1' \in f_{G_1}(m_1, \mathrm{True}C)$ and $m_2' \in f_{G_2}(m_2, \mathrm{True}C)$
2. $g_{G_1 \cup G_2}(m_1 \cup m_2) = g_{G_1}(m_1) \cup g_{G_2}(m_2)$.
3. For each $s \in L(G_1 \cup G_2, m_1 \cup m_2)$, there exist $s_1 \in L(G_1, m_1)$ and $s_2 \in L(G_2, m_2)$ such that $s_1 \leq s$ and $s_2 \leq s$
4. If $C_{out}(G_1) \cap C_{out}(G_2) = \emptyset$, then $L(G_1 \cup G_2, m_1 \cup m_2) = L(G_1, m_1) \cap L(G_2, m_2)$.

Statement 1 in the lemma 2.5 states that if a condition set is sufficient to enable transitions in system $G_1$ to fire to marking $m_1$, and to enable transitions in system $G_2$ to fire to marking $m_2$, then it is sufficient to enable transitions in system $G_1 \cup G_2$ to fire to marking $m_1 \cup m_2$ in the composed system. The converse is also true. Statement 2 in the lemma states that the output of the composed system is just the union of the output of the individual systems. Statement 3 states that for any $C$-sequence $s$ in the language of the composed system, for each of the subsystems there is some $C$-sequence in the subsystem language that is comparable to $s$ and no more descriptive than $s$.

The condition in statement 4 about the composed systems having nonintersecting output condition sets is often true. The lemma statement 4 states that when the output conditions of these individual subsystems are nonintersecting, then the

resulting language of the composed system is just the simple intersection of the individual languages. Figure 2.4 shows an example of a set of subsystems model for a scanner control unit. Dashed arcs in the figure indicate the flow of conditions between subsystems.

## 2.4 Condition System Models Modularity

In this section we will present a special class of condition system models, SPECIFICATION BLOCK (SpecBlock) introduced in [HA98A] & [HGSA00] to demonstrate the modularity of condition models which allow us to standardize, group and reuse system models. The specification block is a condition system used to specify the language that a system model (i.e. an auxiliary system model) should follow. In other words, SpecBlocks specify the desired states of a system model. Following the introduction of the specification block, we will define the properties of SpecBlock by presenting the notion of an $achievable$ SpecBlock and the composition of SpecBlock.

### 2.4.1 Specification Block

We represent our specification of desired system model in terms of a specification block (SpecBlock), defined as a triple $(\mathcal{G}^{SB}, m_{init,SB}, m_{cmpl,SB})$, where $\mathcal{G}^{SB}$ is a condition system, $m_{init,SB} \in M_{\mathcal{G}^{SB}}$ is an INITIATION STATE and $m_{compl,SB} \in M_{\mathcal{G}^{SB}}$ is a COMPLETION STATE. The set of states $M_{\mathcal{G}^{SB}}$ is limited to the set of states reachable from the initiation state.

Next we will characterize the specification block by two condition sets, defined as:

$C_{init,SB} = \{c \in \Phi_{\mathcal{G}^{SB}}(p) \mid p \text{ is marked under } m_{init,SB}\}$

$C_{cmpl,SB} = \{c \in \Phi_{\mathcal{G}^{SB}}(p) \mid p \text{ is marked under } m_{cmpl,SB}\}$

The condition set, $C_{init,SB}$, represents a set of conditions that are true whenever a specification is initiated. Likewise, $C_{cmpl,SB}$ is the set of conditions generated by the final marking within the specification block. We will also defined condition system for system model as $\mathcal{G}^{Sys}$.

In the following definition, we will formally define the completion language of SpecBlock:

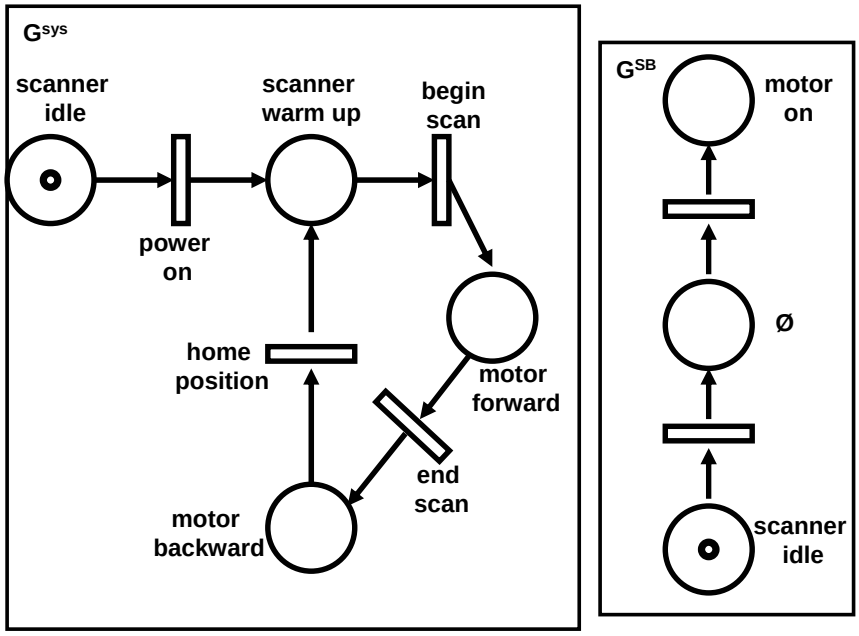


Figure 2.5: An example SpecBlock $\mathcal{G}^{SB}$ for scanner power motor control unit ($\mathcal{G}^{Sys}$) of Figure 2.2

**Definition 2.7** Given a SpecBlock $(\mathcal{G}^{SB}, m_{init,SB}, m_{cmpl,SB})$, the completion language $L_{cmpl}(\mathcal{G}^{SB}) \subseteq L(\mathcal{G}^{SB}, m_{init,SB})$ is defined such that: $s \in L_{cmpl}(\mathcal{G}^{SB})$ if there exists some marking sequence $(m_0 ... m_k)$ consistent with s where $m_0 = m_{init,SB}$ and $m_k = m_{cmpl,SB}$.

We note that the completion language for condition systems is analogous to the marked language in traditional event-based languages. It is not necessary prefix closed.

**Example 2.3** Suppose that the scanner power control unit model shown in Figure 2.2 is the system model, $\mathcal{G}^{Sys}$ and specification block associates with switching the scanner off/on and turning the motor on is shown in $\mathcal{G}^{SB}$ of Figure 2.5. The SpecBlock is represented as $(\mathcal{G}^{SB}, (100), (001))$ where the marking vector correspond to the places from down to up in the figure. One of the C-sequence, s within the completion language is ({Scanner Idle} {$\emptyset$} {Motor On}). The {$\emptyset$} in the C-sequence effectively allows any intermediate activity between the Scanner Idle and Motor On conditions under the descriptive ordering. Note that the C-sequence

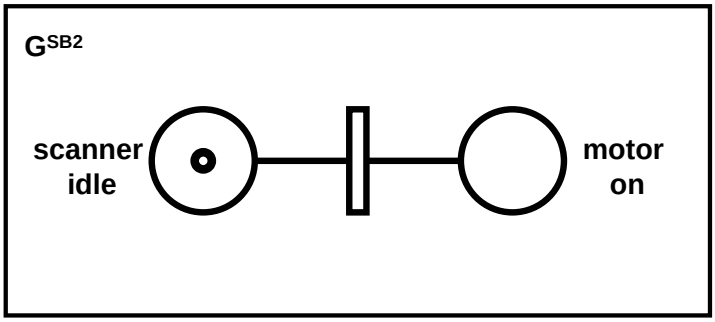


Figure 2.6: An example SpecBlock $\mathcal{G}^{SB2}$ for scanner power motor control unit ($\mathcal{G}^{Sys}$) of Figure 2.2

is ambiguous in the sense that it describes selected output conditions of $\mathcal{G}^{Sys}$, but does not describe details of how to achieve those conditions.

### 2.4.2 Achievable Specification Block

In this subsection, we will now formally define an achievable specification block.

**Definition 2.8** A SpecBlock $(\mathcal{G}^{SB}, m_{init,SB}, m_{cmpl,SB})$ is achievable with respect to a system model $\mathcal{G}^{Sys}$ if for any $m \in M_{\mathcal{G}^{Sys}}$ such that $g_{\mathcal{G}^{Sys}}(m) = g_{\mathcal{G}^{SB}}(m_{init,SB}) \bigcap C_{out}(\mathcal{G}^{Sys})$, then there exists a $s \in L_{cmpl}(\mathcal{G}^{SB})$ and a $s' \in L(\mathcal{G}^{Sys}, m)$ such that $s \leq s'$.

Thus, if a specification $\mathcal{G}^{SB}$ is achievable, then for any state of the system model that matches the output of the initial state of the specblock, there will exist a C-sequence from that state such that the sequence is at least as descriptive as some C-sequence of the completion language of the specblock. Note that ACHIEVABILITY does not imply that the system model could be restricted to just the specified behavior, it simply says that the system model is capable of satisfying the behavior.

**Example 2.4** From example 2.3, the C-sequence within the completion language of $\mathcal{G}^{SB}$, ({Scanner Idle} {$\emptyset$} {Motor On}) is achievable with respect to $\mathcal{G}^{Sys}$ since the system model can produce a C-sequence as descriptive as this. Now suppose we omitted the middle state in the $\mathcal{G}^{SB}$ of Figure 2.5 as shown in Figure 2.6 ($\mathcal{G}^{SB2}$), so that we now had ({Scanner Idle} {Motor On}) as the C-sequence within our com-

pletion language of $\mathcal{G}^{SB2}$, this then would not be achievable with respect to $\mathcal{G}^{Sys}$, since the system model cannot go from scanner idle to motor on without traveling through some intermediate state. C-sequences within the completion language from both $\mathcal{G}^{Sys}$ and $\mathcal{G}^{SB2}$ are said to be incomparable with each other.

### 2.4.3 Composition of Specification Block

Next, we will present results on sequentially composing SpecBlocks into larger SpecBlocks. If each individual SpecBlock is achievable and certain relationships between the SpecBlocks used are satisfied, then the resulting composed block will also be achievable.

We define the SEQUENTIAL COMPOSITION of specification blocks formally in the following definition:

**Definition 2.9 SEQUENTIAL COMPOSITION FOR SPECBLOCK:** Given two SpecBlocks $SB_1 = (\mathcal{G}^{SB_1}, m_{init,SB_1}, m_{cmpl,SB_1})$ and $SB_2 = (\mathcal{G}^{SB_2}, m_{init,SB_2}, m_{cmpl,SB_2})$, define the sequential composition, denoted $SB_1|SB_2 = (\mathcal{G}^{SB_1}|\mathcal{G}^{SB_2}, m_{init,SB_1|SB_2}, m_{cmpl,SB_1|SB_2})$ such that:

1. $\mathcal{G}^{SB_1}|\mathcal{G}^{SB_2}$ is the union of the nets $\mathcal{G}^{SB_1}$ and $\mathcal{G}^{SB_2}$ with the addition of an additional transition $t_{join}$ and arcs such that, arcs lead to $t_{join}$ from each place $p \in \mathcal{P}_{SB_1}$ marked under $m_{cmpl,SB_1}$, and lead from $t_{join}$ to each place $p \in \mathcal{P}_{SB_2}$, marked under $m_{init,SB_2}$.

2. $m_{init,SB_1|SB_2}(p)$ equals $m_{init,SB_1}(p)$ if $p \in \mathcal{P}_{SB_1}$ and equals 0 if $p \in \mathcal{P}_{SB_2}$.

3. $m_{cmpl,SB_1|SB_2}(p)$ equals 0 if $p \in \mathcal{P}_{SB_1}$ and equals $m_{cmpl,SB_2}(p)$ if $p \in \mathcal{P}_{SB_2}$.

Note that the initiation $SB_1|SB_2$ ($\mathcal{G}^{SB,seq}$) is the same as initiating just $SB_1$. When $SB_1$ in the composition reaches completion, then $SB_2$ initiates. The composed block then completes upon the completion of $SB_2$.

The following properties follow from the definition:

**Lemma 2.6 (HA98A)** Consider SpecBlocks $SB_1$ and $SB_2$ is the same as just initiating just $SB_1$. When $SB_1$ in the composition reaches completion, then $SB_2$ initiates. The composed block then completes upon the completion of $SB_2$. The following properties are true:

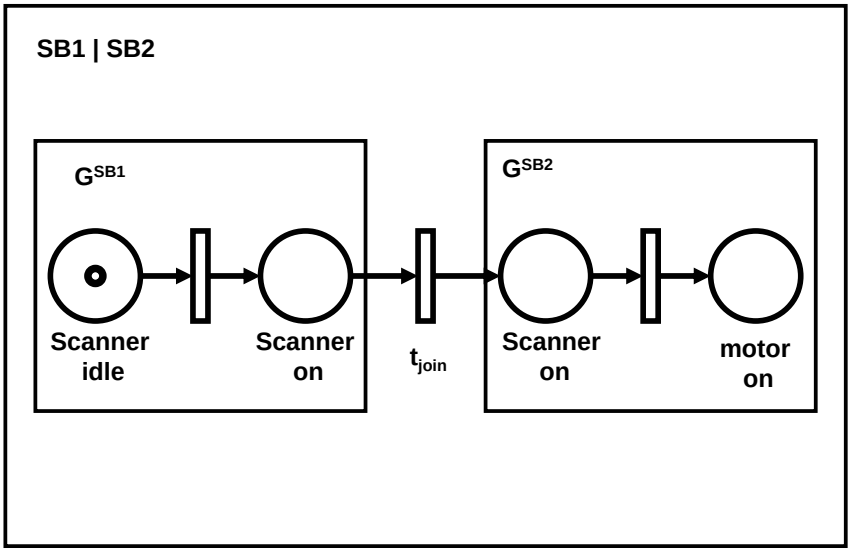


Figure 2.7: An example of Composed SpecBlock SB1|SB2 for scanner power motor control unit ($\mathcal{G}^{Sys}$) of Figure 2.2

1. $C_{init,\mathcal{G}_{SB,seq}} = C_{init,SB_1}$.
2. $C_{cmpl,\mathcal{G}_{SB,seq}} = C_{cmpl,SB_2}$.
3. $L_{cmpl}(\mathcal{G}^{SB,seq}) = \{ss' | s \in L_{cmpl}(\mathcal{G}^{SB_1}), s' \in L_{cmpl}(\mathcal{G}^{SB_2})\}$.

To show the last statement of the lemma, note that for any $s \in L_{cmpl}(G^{SB_1})$, there exists a marking sequence from $m_{init,SB_1}$ to $m_{cmpl,SB_1}$. By the composition, $m_{cmpl,SB_1}$ enables the transition $t_{join}$ between the SpecBlocks. Firing $t_{join}$ empties $G^{SB_1}$ and gives $m_{init,SB_2}$, from which any marking sequence leading to $m_{cmpl,SB_2}$ gives string $s' \in L_{cmpl}(G^{SB_2})$. Since neither SpecBlock has transition condition inputs from the other SpecBlock, then linking of the two blocks does not cause one to restrict the transition firings of the other. By definition 2.8, if follows then that $ss' \in L(\mathcal{G}^{seq,SB}, m_{init,SB_{seq}})$, and $ss' \in L_{cmpl}(SB_{seq})$. We have thus shown that $L_{cmpl}(\mathcal{G}^{SB_{seq}}) \subseteq \{ss' \mid s \in L_{cmpl}(\mathcal{G}^{SB_1}), s' \in L_{cmpl}(\mathcal{G}^{SB_2})\}$ Containment in the other direction is shown by noting that $m_{cmpl,SB_{seq}}$ is $m_{cmpl,SB_2}$ by its definition, and since $\mathcal{G}^{SB_2}$ is initially unmarked in the composition, no marking sequence can reach $m_{cmpl,SB_{seq}}$ without first reaching $m_{cmpl,SB_1}$ to enable the firing of $t_{join}$.

**Example 2.5** Consider the composed Specblock SB1|SB2 shown in Figure 2.7, $SB1 = (\mathcal{G}^{SB1}, (10), (01))$ is achievable with respect to system model of Figure 2.2

$G_{Sys}$ and $SB2 = (\mathcal{G}^{SB2}, (10), (01))$ is also achievable with respect to $\mathcal{G}^{Sys}$. SB1|SB2 will be achievable too w.r.t $\mathcal{G}^{Sys}$ since $C_{init,\mathcal{G}^{SB_2}} = \{Scanner\ on\}$, $C_{cmpl,\mathcal{G}^{SB_1}} = \{Scanner\ On\}$ and therefore $C_{init,\mathcal{G}^{SB_2}} \subseteq C_{cmpl,\mathcal{G}^{SB_1}}$. But SB2|SB1 will not be achievable w.r.t. $\mathcal{G}^{Sys}$ since $C_{init,\mathcal{G}^{SB_1}} \not\subseteq C_{cmpl,\mathcal{G}^{SB_2}}$.

Finally we will present the main result of achievable sequential composition as presented in lemma 2.7:

**Lemma 2.7 (HA98A)** Given SpecBlocks $SB_1$ and $SB_2$ which are achievable w.r.t. $\mathcal{G}^{Sys}$, the sequential SpecBlock composition, $SB_1|SB_2$, is achievable w.r.t. $\mathcal{G}^{Sys}$ if $C_{init,\mathcal{G}^{SB_2}} \subseteq C_{cmpl,\mathcal{G}^{SB_1}}$.

# Chapter 3

# Modeling For Interfacing

Modeling for interfacing focuses on issues in simulating environment interfaces with a particular target system. The environment subjected to simulation communicates with target system through a common interface and the objective of modeling for interfacing is to build a system model, $\mathcal{G}^{\text{sys}}$ out of the environment solely for the objective of imitating its interaction with the target system. In other words, the system model created is only responsible for providing an accurate system interaction with its target system. A successful modeling process will lead target system into "believing" that it is interacting with the actual system even though the system it interacted with is merely a virtual system and it is not the actual and original system. Under the modeling for interfacing framework, a subsystem model will then be synthesized from $\mathcal{G}^{\text{sys}}$ to perform fault detection on signal interactions between target system and $\mathcal{G}^{\text{sys}}$. The concept of modeling for interfacing will serve as a fundamental foundation for the introduction of two special classes of condition system model, virtualblock and detectblock.

In our cases, we are given a system under test (SUT) which interacts primarily with an auxiliary system and a controller. The SUT will be our target system and the auxiliary system will be the system model. Controller's function is to determine whether SUT is in working condition and the auxiliary system will act as a supplement to SUT through their signals interface interaction. In addition, the specification of SUT is only partially known or understood, and therefore a complete system model of SUT is unavailable. However, we will have the necessary specification of the auxiliary system which will allow us to develop the corresponding system model. Under our modeling methodologies, auxiliary system, the system interfacing with system under test, is subjected to modeling process and is represented as a form of condition system Petri net. The auxiliary system model will not only provide signal interactions with SUT, but also detect faults on SUT.

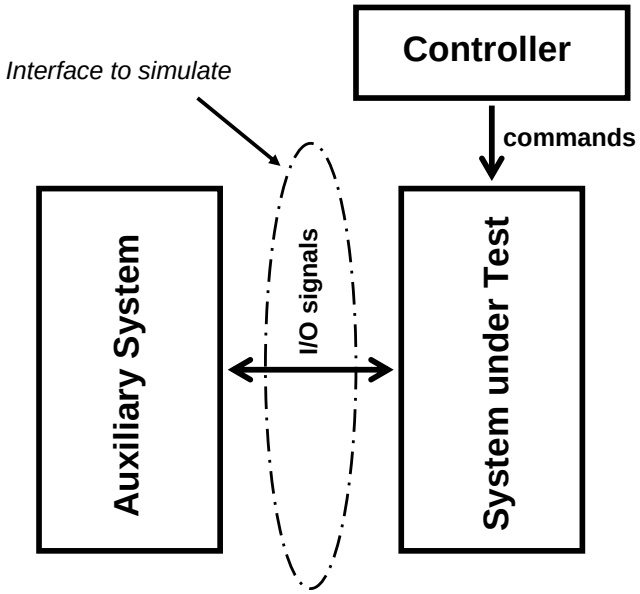


Figure 3.1: System interactions among system under test, auxiliary system and controller

One of the goals of this thesis is to develop a system model out of the real and actual auxiliary system for the purpose of interfacing with system under test. Motivations for developing such system model originate from the disadvantages of using a real system in terms of design time, manufacturing cost and also the potential danger involved in actual system testing. By using a system model (such as a software model), the corresponding system behavior can either be operated as a STANDARD BEHAVIOR where both "normal" and "rare" behavior can be simulated or CUSTOMIZED BEHAVIOR where forced behavior such as peripheral failures and errors can be modeled. In comparison to hard coding specific responses, a model based approach is more suitable in light of flexibility on future testing and design. Coding can be synthesized from the model and device modeling can be done with multiple behaviors instead of just a single response. Thus in order to test certain functionality of the SUT, we will utilize the model-based approach to model the auxiliary system.

The goal of the auxiliary system modeling is to eventually confirm the SUT correct signals interaction with the auxiliary system. To achieve this goal, there are mainly two tasks to be completed: 1. Given an excitation of auxiliary system by

SUT, determine the possible responses of auxiliary system. 2. Given the possible responses of the auxiliary system, confirm that the SUT responds appropriately (fault detection). The first task will be accomplished by virtualblock which will be addressed in the following chapter, chapter 4. Another class of condition system model, detectblock is introduced to tackle the second task. Detectblock will be addressed in chapter 5.

This chapter is presented as follow: we will first briefly define the notion of real and expected behavior on a system. From such notions, we will then formally define our definition of fault. Next, we will present and define the system architecture under our modeling for interfacing framework which mainly consist of: System Under Test, Auxiliary System, Controller and External Environment. Finally, fault detection under modeling for interfacing framework will be defined and described in the last section of the chapter.

## 3.1 Real and Expected Systems

In this section, we will define the notion of real and expected behavior of a system presented in [A04]. For a given system, we use superscripts (R,E) to distinguish between the real and actual behavior of a system, $\mathcal{G}^R$ and expected behavior of a system model, $\mathcal{G}^E$. The REAL SYSTEM distinguishes itself from the EXPECTED SYSTEM through the definition of fault which we will briefly define in Definition 3.1:

**Definition 3.1** Under the notion of real and expected behavior, a system is said to have a FAULT if the language of the real system ($\mathcal{G}^R$) is not contained within the language of its corresponding model system ($\mathcal{G}^E$) of the expected behavior, i.e. For all $s \in L(\mathcal{G}^R, m_0^R)$, if $s \notin L(\mathcal{G}^E, m_0^E)$ then s represents a fault.

Ideally, an expected system completely captures the necessary system behavior of a real system and therefore it is fault free. The expected behavior of a system model is considered as a subset of the real system, $\mathcal{G}^E \subseteq \mathcal{G}^R$.

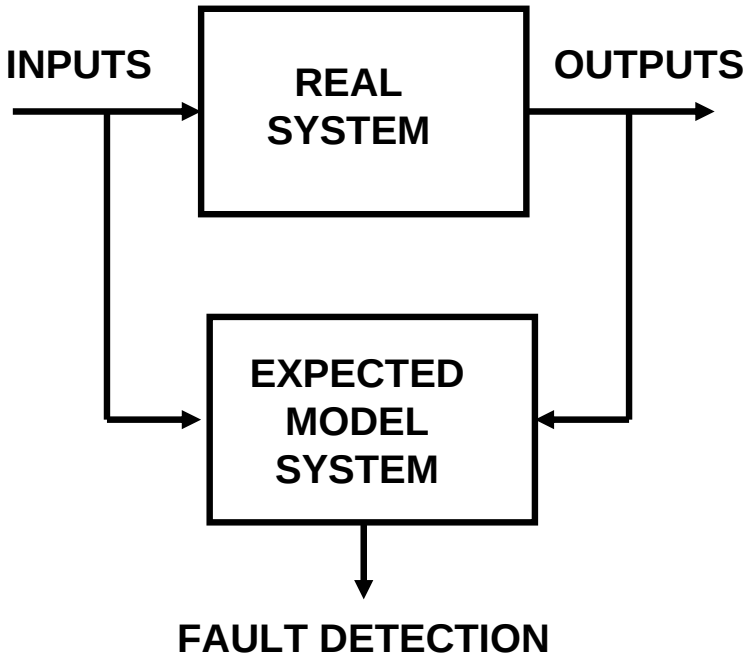


Figure 3.2: Scheme for Real and Expected System Fault Detection

## 3.2 Systems under Modeling for Interfacing Framework

In general, there are four systems we consider: SYSTEM UNDER TEST, AUXILIARY SYSTEM, CONTROLLER and EXTERNAL ENVIRONMENT. For any given system, we use superscripts (SUT, Aux) to distinguish between System under Test, $\mathcal{G}^{\text{SUT}}$ and Auxiliary System, $\mathcal{G}^{\text{Aux}}$. Superscripts (Ctrl, Env) will be used to define the Controller, $\mathcal{G}^{\text{Ctrl}}$ and the External Environment, $\mathcal{G}^{\text{Env}}$. We will also use superscript (I) to denote system which is responsible for interfacing any target systems, $\mathcal{G}^{\text{I}}$ and superscript (D) to denote system which is capable of performing fault detection, $\mathcal{G}^{\text{D}}$.

In our cases, system under test and auxiliary system are the principal systems under the modeling for interfacing framework. Each of them interacts with one another through some common signal paths. SUT will communicate with auxiliary system through an I/O interface by issuing input signals to auxiliary system and receiving output signals from the auxiliary system. Under our modeling methodologies, a real system under test is interacting with an expected auxiliary system which is fault-free and the expected auxiliary system will have an additional capability of detecting a fault on SUT.

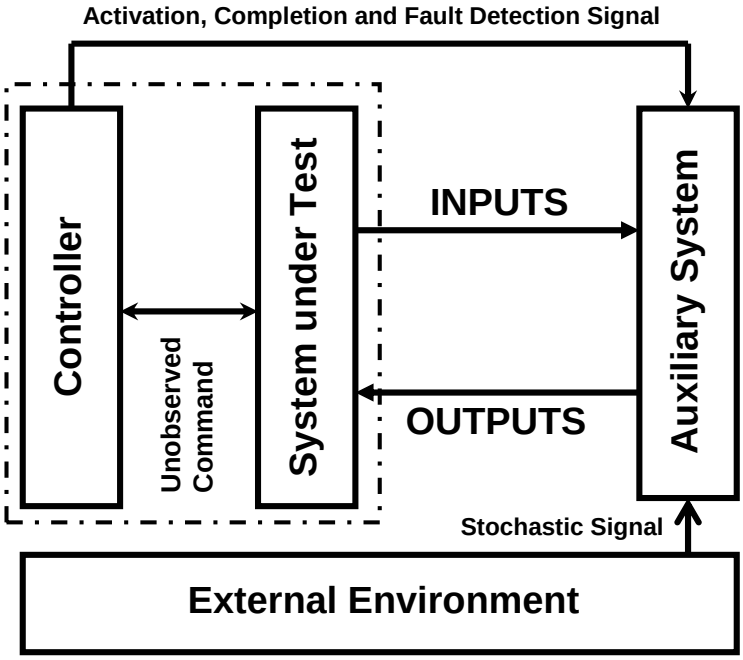


Figure 3.3: Scheme for General System within the Framework of Modeling for Interfacing

The controller is included in our framework due to the fact that each signal cycle between SUT and auxiliary system required an independent indicator. The controller will communicate with both SUT and auxiliary system indicating the start or the end of a particular command cycle. As stated before, the function of the controller is to act as a supervisor for SUT and determine the functionality of SUT in each cycle of commands but its commands to SUT are unobservable with respect to auxiliary system. The controller will interact with auxiliary system to signal the activation of each SUT command cycle and also the completion of the corresponding command cycle. Also note that the controller is assumed to be fault-free for the convenience of fault detection on SUT.

Another system we consider under the framework of modeling for interfacing is the external environment. The external environment will represent the real world surrounding systems within the framework and it is the source of stochastic/random signals to systems (typically the auxiliary system). The inclusion of external environment in the framework is essential to the modeling process of auxiliary system.

Figure 3.3 further illustrates the systems within the modeling for interfacing framework and their relationships with each others. We will formally define the systems under our modeling framework in the following definition:

**Definition 3.2** Systems considered within the framework of modeling for interfacing are real system under test, ($\mathcal{G}^{SUT,R}$); expected auxiliary system with system interfacing and fault detecting capabilities, ($\mathcal{G}^{Aux,E,I,D}$); the controller ($\mathcal{G}^{Ctrl}$) and the external environment ($\mathcal{G}^{Env}$). Detailed descriptions for each systems are given below:

1. $\mathcal{G}^{SUT,R}$ is the system representing the real behavior of system under test and it is possible to have some faulty behaviors within the system.

2. $\mathcal{G}^{Aux,E,I,D}$ is the system representing the expected behavior of auxiliary system which provides I/O signals interaction with system under test and it will perform fault detection on system under test.

3. $\mathcal{G}^{Ctrl}$ is the system representing the supervisor of system under test. It is responsible for issuing activation signal $c_{init,ctrl}$, completion signal $c_{cmpl,ctrl}$ and also the fault detection signal $c_{d,ctrl}$ into auxiliary system.

4. $\mathcal{G}^{Env}$ is the system representing the external environment surrounding the auxiliary system, $\mathcal{G}^{Aux}$. The output conditions of the system $C_{out}(\mathcal{G}^{Env})$ are non-deterministic with respect to $\mathcal{G}^{Aux}$.

There are basically two different kinds of SUT: Real and Expected. One of the main differences between a real and an expected SUT is that, for the real system there is a possibility that faulty behaviors might exist within its system, whereas for the expected system the probability of fault occurrence is zero since the expected system completely captures the necessary system behaviors of system under test. In order to perform fault detection on SUT, we will therefore use a real SUT in our modeling framework.

An auxiliary system is the simulating environment supplemental to system under test. As stated before, the tester has the necessary information to model and simulate the system behavior of auxiliary system which allows us to extract an expected system out of the auxiliary system. The expected auxiliary system is modeled such that it is assumed to be free of faulty system behavior. There are generally two

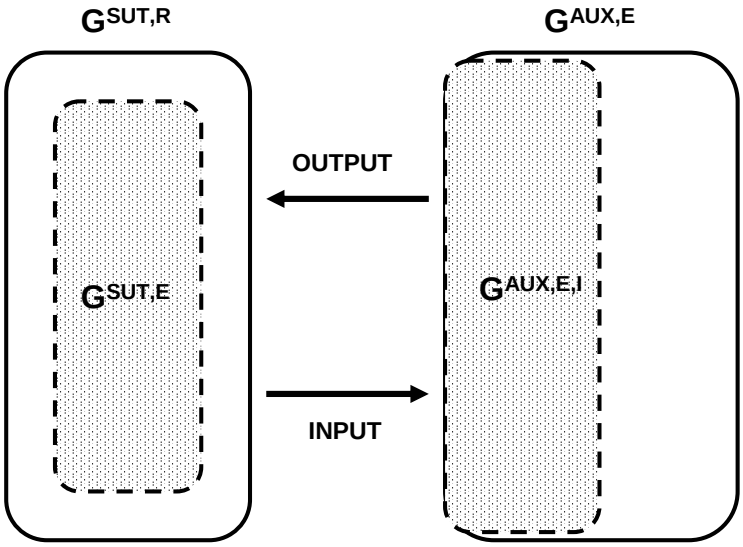


Figure 3.4: SUT and Auxiliary System Interfacing under Modeling for Interfacing Framework

different kinds of expected auxiliary systems we considered: the expected model, $\mathcal{G}^{\mathrm{Aux,E}}$ and the expected model for interfacing SUT, $\mathcal{G}^{\mathrm{Aux,E,I}}$. The expected model is the expected auxiliary system which consist of all the necessary system behavior of auxiliary system whereas the expected model for interfacing SUT is the expected system which consist of just the system behavior of auxiliary system necessary for the purpose of interfacing with SUT. These two expected models however are not sufficient to meet the goals of our modeling framework. We need to synthesize a fault detector from the expected auxiliary system to perform the testing of SUT. The fault detector will be incorporated into an expected auxiliary system and we will denote such system as $\mathcal{G}^{\mathrm{Aux,E,I,D}}$.

We define the system which initiates and controls the SUT and auxiliary system as the controller. The controller acts like an external supervisor or PC which will supervise the signals interaction between the auxiliary system and SUT. Initiation condition signal, $\mathrm{c}_{\mathrm{init,ctrl}}$ from the controller will activate the beginning of command cycle between SUT and the auxiliary system by telling auxiliary system to begin receiving input signals from SUT. At the end of the command cycle the controller will issue a completion condition signal, $\mathrm{c}_{\mathrm{cmpl,ctrl}}$ to auxiliary system to

indicate that it is time to provide output signals to SUT. In addition, the controller of modeling for interfacing framework will also issue a fault detection signal, $c_{d,ctrl}$ to $\mathcal{G}^{Aux,E,I,D}$ to perform fault detection on I/O interface between SUT and auxiliary system at the end of every commands cycle.

Finally the system which interacts externally within the auxiliary system is called the external environment according to statement 4. The system is analogous to the real world where it is the main source of non-deterministic events for the auxiliary system. Its input to auxiliary system is unpredictable and therefore it is incompatible with our modeling framework which is deterministic. Under our modeling methodologies, the random signals interaction between the external environment and auxiliary relationship will be reduced to a fixed and deterministic signals interaction. Such relationship with the auxiliary system is vital for our modeling process especially in the case of expected auxiliary system interfacing SUT. Recall that the expected auxiliary system interfacing SUT is the subsystem of expected auxiliary system model and there will be some system components in expected auxiliary system that are not being modeled. Such system components might be responsible for interacting with the external environment and without the appropriate subsystem model we will not be able to obtain their system dynamics. Therefore the predefined, fixed, deterministic external environment incoming signals assumptions will be made to overcome the missing system interactions issue. Such assumption is necessary for the success of our modeling process and also will not affect the realness of our auxiliary system model.

## 3.3 Fault Detection under Modeling for Interfacing Framework

Next, we will define fault detection on system under modeling for interfacing framework by exploring the systems relationship among different subsystems of auxiliary systems and systems under test. The systems relation between auxiliary system and system under test are presented in term of real, expected system behavior and system with detector capability.

First of all, for $\mathcal{G}^{SUT,E}$ and $\mathcal{G}^{Aux,E}$, this is the perfect ideal situation where no faults are possible. Both $\mathcal{G}^{SUT,E}$ and $\mathcal{G}^{Aux,E}$ represent the expected system behaviors of system under test and auxiliary system, thus there will be no faulty system behavior

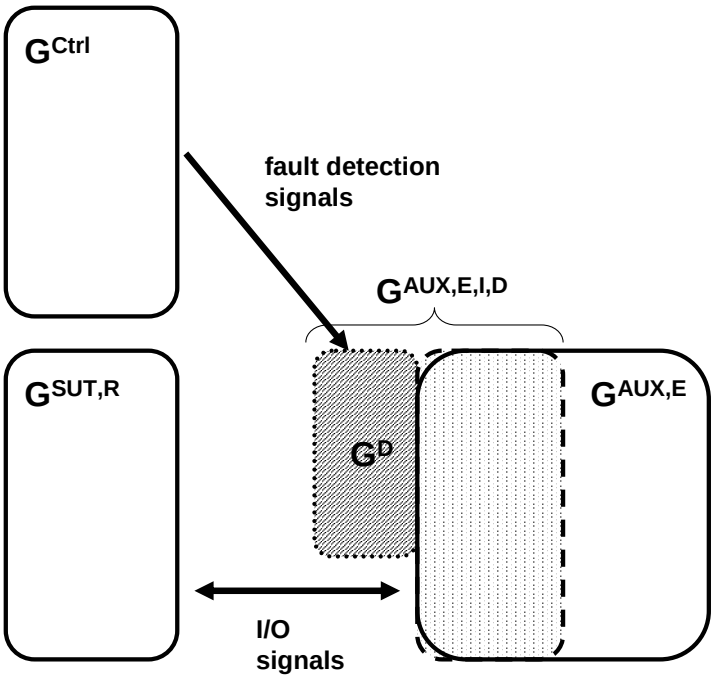


Figure 3.5: Fault Detection on SUT under Modeling for Interfacing Framework

in existence.

Secondly, for $\mathcal{G}^{SUT,E}$ and $\mathcal{G}^{Aux,E,D}$, this is the situation where $\mathcal{G}^{SUT,E}$ is fault free and the expected auxiliary system with detector capability $\mathcal{G}^{Aux,E,D}$ will never detect a fault. $\mathcal{G}^{Aux,E,D}$ consists of fault detector that will perform testing on $\mathcal{G}^{SUT,E}$ and always return a negative fault detection signal because $\mathcal{G}^{SUT,E}$ represent system with expected system behavior and thus it is fault free.

Finally, for $\mathcal{G}^{SUT,R}$ and $\mathcal{G}^{Aux,E,D}$, this is the situation where $\mathcal{G}^{SUT,R}$ represent the real and actual system and faulty system behaviors are possible and therefore it is possible for expected auxiliary system with detector capability, $\mathcal{G}^{Aux,E,D}$ to detect a fault. The real and actual system under test, $\mathcal{G}^{SUT,R}$ is subjected to faulty system behavior and auxiliary system with detector, $\mathcal{G}^{Aux,E,D}$ will have the capability to detect it. This systems relationship is the central of this thesis, and these are the systems that are being considered under our modeling process and where fault detection methodologies is being applied.

# Chapter 4

# Virtualblocks

From the general perspective on systems within the modeling for interfacing framework, we will now shift our attention to one of the building blocks of the framework, the VIRTUALBLOCK. Virtualblock is a special class of condition system model responsible for providing a virtual system to a specific target system (i.e. a SUT) as described in chapter 3. It is the vital element of our modeling for interface framework which provides the simulation environment to SUT and can also be known as a subsystem of auxiliary system model which interfaces with SUT, $\mathcal{G}^{Aux,E,I}$. Virtualblocks contain two special class of condition system models: INPUTBLOCK and OUTPUTBLOCK. The role of the inputblock is to recognize incoming signals from the target system according to a specific pre-defined specification of the I/O interactions between target system and system model whereas the function of outputblock is to produce appropriate outgoing signals from the system model to the target system according to the specification of system model.

In this chapter, we will explore the system properties of virtualblock by first presenting the notion of LEGALITY. Next in section 4.2, we will formally define one of the system components of virtualblock, the inputblock. Following the formal definition of inputblock, formal definition of another system component of virtualblock, the outputblock will also be presented in section 4.3. In section 4.4, we illustrate the composition of inputblock and outputblock which will lead to the formation of virtualblock. Sequential composition of inputblocks is then defined to model clocked signal in discrete time environment. Finally, in the last section of the chapter, inputblock's construction procedures for identifying incoming target system signal will be presented. Construction procedures of outputblock for outputting outgoing system model signal to target system will be presented in the similar manner. Both of the construction procedures for inputblock and outputblock explore different modeling techniques on clocked and non-clocked signals. We conclude the chapter by

presenting construction procedure for composition of multiple inputblocks & outputblocks and also the activation of virtualblocks by the Controller.

## 4.1 Legality

In this section, we will present a brief overview of legality over condition signals and illustrate the legal language which describes the permitted system behavior of a condition system. A C-sequence is said to be within a particular legal language if it is permitted by the corresponding predefined rules/orders of a system. Let $C_{legal} \subset AllC$ be a set of conditions which is legal, this will imply that the condition set is within a legal C-sequence, $s_{legal}$. The sequence of legal condition sets, $s_{legal} \in L_{legal}(\mathcal{G})$ consequently will be used to represent the allowable system behavior of the condition system $\mathcal{G}$. Generally a legal C-sequence $s_{legal}$ can either be the inputs to $\mathcal{G}$ or outputs of $\mathcal{G}$, $s_{legal} \subseteq s\ |_{C_{in}(\mathcal{G})}$ or $s_{legal} \subseteq s\ |_{C_{out}(\mathcal{G})}$. In some cases, a legal C-sequence $s_{legal}$ can also represent both input and output signal of a system.

## 4.2 Inputblock

Next, we will define the system properties of inputblock. Note that the system properties of inputblock is similar to specification block presented in Chapter 2. The general structure of condition system model for inputblock is shown in Figure 4.1. The box in Figure 4.1 contains different kinds of condition model nets and their structure will depend upon types of incoming signals the inputblock is detecting. Despite the difference, every box will at least consist of a transition denoted as $t_{legal}$. We will formally define inputblock in the following definition:

**Definition 4.1** Inputblock is a form of condition system model defined as a 4-tuple $(\mathcal{G}^{IB}, m_{init,IB}, m_{legal,IB}, s_{legal,IB})$, where $\mathcal{G}^{IB}$ is a condition system, $m_{init,IB} \in M_{\mathcal{G}^{IB}}$ is an INITIATION STATE, $m_{legal,IB} \in M_{\mathcal{G}^{IB}}$ is a LEGAL STATE. Marking $m_{init,IB}$ will go to marking $m_{legal,IB}$ if and only if sequence $s_{legal,IB}$ is received.

In our cases, $s_{legal,IB}$ will be the sequence of conditions that is true whenever the corresponding condition set is within a legal incoming C-sequence signals from the target system which represent the permitted system behavior of system model. Under our modeling for interfacing framework, $s_{legal,IB}$ will be viewed as incoming

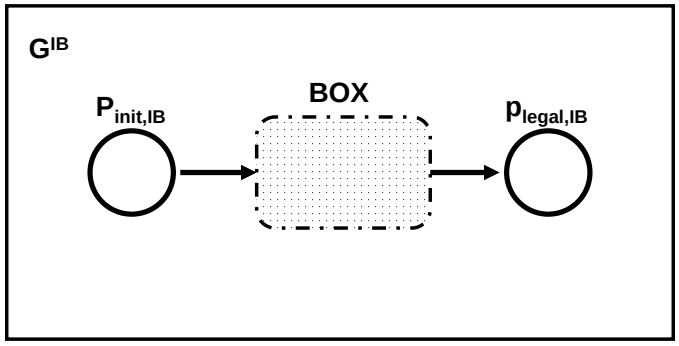


Figure 4.1: General Structure of Inputblock

C-sequence from SUT. This sequence can either be clocked or unclocked as we discuss later. Finally as the last part of this section, we will formally define the legal language of inputblock in the following definition:

**Definition 4.2** Given an inputblock $(\mathcal{G}^{\mathrm{IB}}, m_{init,\mathrm{IB}}, m_{legal,\mathrm{IB}})$, the legal language of the inputblock $L_{legal}(\mathcal{G}^{\mathrm{IB}}, m_{init,\mathrm{IB}}, m_{legal,\mathrm{IB}})$ represents the permitted system behavior of the system under modeling. It is defined such that for all $s \in L(\mathcal{G}^{\mathrm{IB}})$ where k is some positive integer, if there exists a marking sequence $(m_0 ... m_k)$ of s satisfies these conditions: $m_0 = m_{init,\mathrm{IB}}$ and $m_k = m_{legal,\mathrm{IB}}$ then $s \in L_{legal}(\mathcal{G}^{\mathrm{IB}}, m_{init,\mathrm{IB}}, m_{legal,\mathrm{IB}})$.

Note that in contrast to inputblock, $m_{legal,\mathrm{IB}}$ is not the final marking of virtualblock and identifying the incoming legal language is only part of the functions of virtualblock.

## 4.3 Outputblock

In this section, we will formally define the OUTPUTBLOCK. The general structure of condition system model for outputblock is shown in Figure 4.2 which is very similar to inputblock's model. The box in Figure 4.2 also contains different kinds of condition nets and their structure will depend upon the types of outgoing signals outputting from auxiliary system model to SUT. Every box of outputblocks will at least consist of a transition denoted as $t_{cmpl,ctrl}$. We will formally define outputblock in the following definition:

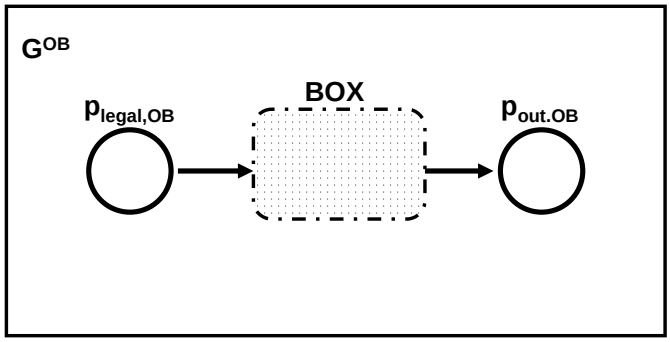


Figure 4.2: General Structure of outputblock

**Definition 4.3** Outputblock is a form of condition system model defined as a 4-tuple $(\mathcal{G}^{OB}, m_{legal,OB}, m_{out,OB}), s_{legal,OB}$, where $\mathcal{G}^{OB}$ is a condition system, $m_{legal,OB} \in M_{\mathcal{G}^{OB}}$ is a LEGAL STATE and $m_{out,OB} \in M_{\mathcal{G}^{OB}}$ is an OUTPUT STATE. The system under $m_{legal,OB}$ as initial state can move to marking $m_{out,OB}$, and in doing so will generate output C-sequence $s_{legal,OB}$. There also exists transition $t_{cmpl,ctrl}$ such that $C_{cmpl,ctrl} = \Phi_{\mathcal{G}^{OB}}(t_{cmpl,ctrl})$.

The condition set, $C_{cmpl,ctrl}$, represents a set of conditions that is true whenever the Controller issue the completion signal to system model to signal the end of the corresponding command cycle. $s_{legal,OB}$ is the set of conditions generated by the system evolving to the final marking within the virtualblock. $s_{legal,OB}$ is responsible for outputting legal condition signals to the target system. Under our modeling framework, $C_{cmpl,ctrl}$ is a singleton condition set since there is only one controller condition signal associated with it and $s_{legal,OB}$ will be regarded as the outgoing C-sequence to SUT. Legal language for outputblock will be described in next definition.

**Definition 4.4** Given an outputblock $(\mathcal{G}^{OB}, m_{legal,OB}, m_{out,OB})$, the legal language of the outputblock $L_{legal}(\mathcal{G}^{OB}, m_{legal,OB}, m_{out,OB})$ represents the permitted system behavior of the system under modeling. It is defined such that for all $s \in L(\mathcal{G}^{OB})$ where k is some positive integer, if the marking sequence $(m_0 ... m_k)$ of s satisfies these conditions: $m_0 = m_{legal,OB}$ and $m_k = m_{out,OB}$ then $s \in L_{legal}(\mathcal{G}^{OB}, m_{legal,OB}, m_{out,OB})$.

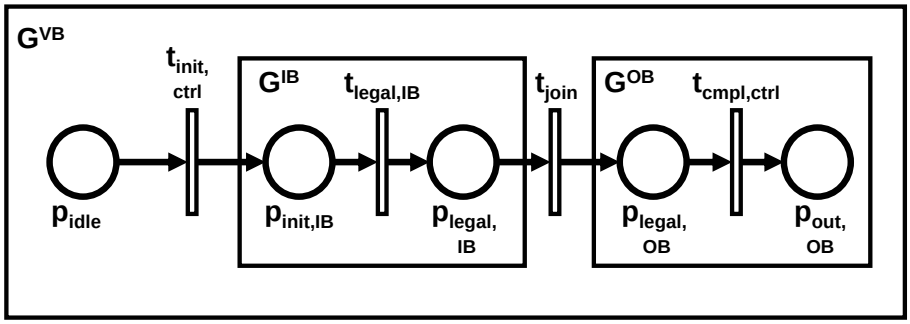


Figure 4.3: Example of Virtualblock

Marking $\mathrm{m}_{\mathrm{out,OB}}$ will be the final marking of outputblock and ultimately the final marking of virtualblock. Upon the successful generation of the marking $\mathrm{m}_{\mathrm{legal,OB}}$, virtualblock would wait for the completion signal from the Controller to generate the final marking/state $\mathrm{m}_{\mathrm{out,OB}}$. The generation of marking $\mathrm{m}_{\mathrm{out,OB}}$ will indicate the end of the virtualblock's task for the corresponding command cycle.

## 4.4 Composition of Inputblocks and Outputblocks

The composition of inputblock & outputblock will form a virtualblock. The purpose of virtualblock composition is to identify legal language and also output legal language for each signals cycle. Example for composition of inputblock and outputblock to create a condition net of virtualblock is shown in Figure 4.3. Be reminded that the inputblock and outputblock shown in the figure are specific examples from general condition net shown in Figure 4.1 & 4.2. We will formally define virtualblock in the following definition:

**Definition 4.5** Virtualblock is a form of condition system model defined as a doublet $(\mathcal{G}^{\mathrm{VB}}, \mathrm{m}_{\mathrm{idle,VB}})$, where $\mathcal{G}^{\mathrm{VB}}$ is a condition system and $\mathrm{m}_{\mathrm{idle,VB}} \in \mathrm{M}_{\mathcal{G}^{\mathrm{VB}}}$ is an IDLE STATE. $\mathcal{G}^{\mathrm{VB}}$ is the union of $\mathcal{G}^{\mathrm{IB}}$ and $\mathcal{G}^{\mathrm{OB}}$ with the addition of one additional place $\mathrm{p}_{\mathrm{idle}}$ and two additional transitions $\mathrm{t}_{\mathrm{init,ctrl}}$ and $\mathrm{t}_{\mathrm{join}}$ such that arcs lead from place marked under $\mathrm{m}_{\mathrm{idle,VB}}$, $\mathrm{p}_{\mathrm{idle}}$ to $\mathrm{t}_{\mathrm{init,ctrl}}$ and then from $\mathrm{t}_{\mathrm{init,ctrl}}$ to place $\mathrm{p}_{\mathrm{init,IB}} \in \mathcal{P}_{\mathrm{IB}}$. Another arc lead from place $\mathrm{p}_{\mathrm{legal,IB}} \in \mathcal{P}_{\mathrm{IB}}$ to transition $\mathrm{t}_{\mathrm{join}}$ and from $\mathrm{t}_{\mathrm{join}}$ to place $\mathrm{p}_{\mathrm{legal,OB}} \in \mathcal{P}_{\mathrm{OB}}$. Virtualblock can be characterized by a condition sets, defined as: $\mathrm{C}_{\mathrm{init,ctrl}} = \Phi_{\mathcal{G}^{\mathrm{VB}}}(\mathrm{t}_{\mathrm{init,ctrl}})$ where $\mathrm{t}_{\mathrm{init,ctrl}}$ is the input to place marked under $\mathrm{m}_{\mathrm{init,IB}}$ of inputblock.

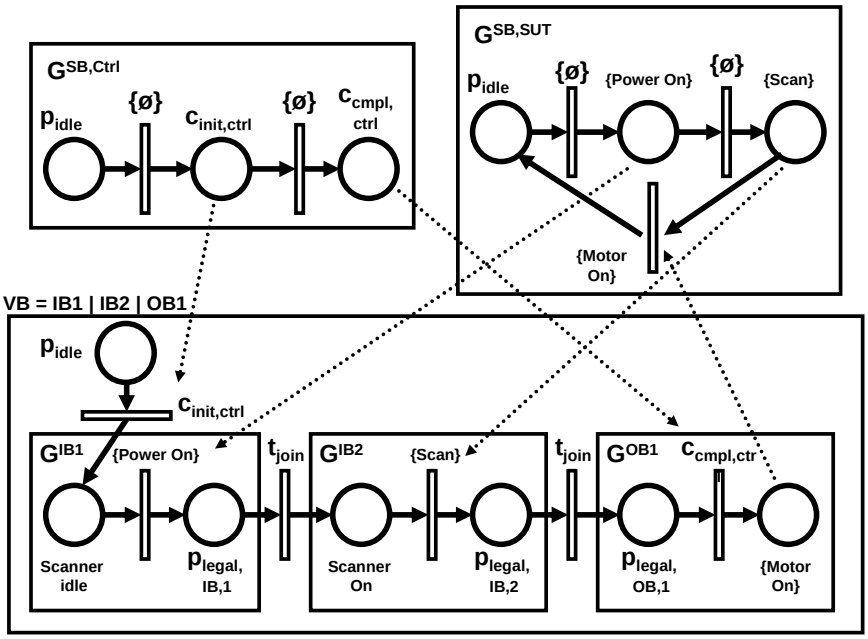


Figure 4.4: Example of Sequential Composition for Two Inputblocks in Virtualblock interacts Specblocks with SUT and the Controller

We define $C_{init,ctrl} = \Phi_{\mathcal{G}^{VB}}(t_{init,ctrl})$. The condition set $C_{init,ctrl}$ as noted in Chapter 3 will be enabled whenever the Controller issue the activation signal to system model to indicate the start of command cycle and it represents the set of conditions responsible for generating the initial marking within the inputblock. We also define $\Phi_{\mathcal{G}^{VB}}(t_{join}) = \{\emptyset\}$.

### 4.4.1 Sequential Composition of Inputblocks

Due to clocking issue in clocked signal, single inputblock as shown in Figure 4.1 is only suitable for identifying continuous time incoming signal from the target system. For identifying incoming clocked signals in discrete time scenario, we will require the use of multiple inputblocks composition. Instead of using concurrent composition, sequential composition will be used to model and compose the group of inputblocks. We will define the sequential composition of inputblocks formally in the following definition:

**Definition 4.6 SEQUENTIAL COMPOSITION FOR INPUTBLOCKS :**

Given two inputblocks $IB_1 = (\mathcal{G}^{IB_1}, m_{init,IB_1}, m_{legal,IB_1})$ and $IB_2 = (\mathcal{G}^{IB_2}, m_{init,IB_2}, m_{legal,IB_2})$ define the sequential composition, denoted $IB_1|IB_2 = (\mathcal{G}^{IB_1}|\mathcal{G}^{IB_2}, m_{init,IB_1}, m_{init,IB_2}, m_{legal,IB_1}, m_{legal,IB_2})$ such that: $\mathcal{G}^{IB_1}|\mathcal{G}^{IB_2}$ is the union of the nets $\mathcal{G}^{IB_1}$ and $\mathcal{G}^{IB_2}$ with the addition of an additional transition $t_{join}$ and arcs such that, arcs lead to $t_{join}$ from place $p \in \mathcal{P}_{IB_1}$ marked under $m_{legal,IB_1}$, and lead from $t_{join}$ to place $p \in \mathcal{P}_{IB_2}$, marked under $m_{init,IB_2}$.

Note that place marked $m_{init,IB_1}$ will also be denoted as $p_{init,VB}$ during the virtualblock formation. Next we will present an example for sequential composition of two inputblocks in system interaction among virtualblock(auxiliary model), SUT and the Controller as shown in Figure 4.4.

**Example 4.1** Consider the scanner power control unit $\mathcal{G}^{sys}$ described in Example 2.3 of Chapter 2. We will model the $\mathcal{G}^{sys}$ as a virtualblock composed of two inputblocks and one outputblock. The task of legal language identification of virtualblock is designated through the sequential composition of inputblocks. Each inputblocks is responsible for recognizing the incoming signal $\{PowerOn\}$ and $\{Scan\}$ from the specblock of SUT ($\mathcal{G}^{SB,SUT}$) respectively and the outputblock will communicate with SUT($\mathcal{G}^{SB,SUT}$) by outputting the outgoing signal $\{MotorOn\}$ to $\mathcal{G}^{SB,SUT}$ to indicate the end of command cycle. The activation and completion of the command cycle are controlled by the specblock of the Controller ($\mathcal{G}^{SB,Ctrl}$) through the issuance of signal $\{c_{init,ctrl}\}$ & $\{c_{cmpl,ctrl}\}$. The controller is mutually independent from auxiliary system & SUT and its condition net in Figure 4.4 is designed for a single command cycle operation between auxiliary system and SUT.

The legal identification of virtualblock can ultimately be accomplished by a single inputblock. The modeling technique to create such inputblock will be discussed in the next section and the inputblocks presented in Example 4.1 is meant for demonstrating the sequential composition of inputblocks. Sequential composition of outputblocks within a virtualblock is not required since outputblock only works in continuous time environment and a single outputblock is sufficient to complete the task of legal language outputting.

## 4.5 Algorithms

From section 4.2, 4.3 and 4.4, we have separately defined the inputblock, outputblock, virtualblock and presented their general condition net structure in high level system perspective. In this section, we will explore the modeling techniques within inputblock and outputblock to create specific condition net structure for identifying and outputting legal languages within two different types of incoming signals: clocked and non-clocked. We will also present the modeling techniques for composing multiple virtualblocks to form a complete system model.

### 4.5.1 Construction Procedures for Inputblocks

In this subsection, we will present algorithm 4.1 & 4.2 to describe how we perform legality identification within the auxiliary system model. These algorithms are based on the clocking and non-clocking properties of incoming signal from SUT. Legal language of inputblock can be identified through the specification of the auxiliary system. Given the specification, we can associate each input command with C-sequence signal. Therefore for an input command with a singular signal, we will associate the command with a C-sequence of single condition signal $s = (\{c\})$. Sometimes an input command will compose of several singular signals in parallel. In that case, we will assign such command with a C-sequence of single condition set $s = (C)$.

For an input command compose of multiple singular signals in serial, the legal language will consist of a C-sequence with multiple singular condition signal and/or condition sets $s = (C_0...C_M)$ where M is the total number of condition sets in the C-sequence and some of these condition sets may be singletons. In addition to that the C-sequence is also sequence oriented. It means that the order of the elements in the C-sequence will also be accounted towards the legality of the entire command. Such input command can also be known as incoming sequential signal from SUT. There are also basically two types of sequential signal, clocked and non-clocked . For non-clocked sequential signal, the corresponding C-sequence will not have a condition signal governing the clocking of its elements, whereas for clocked sequential signal, there will be a condition signal that is responsible for synchronizing the clocking of its elements. We will formally define the condition signal synchronization of C-sequence in the following definition.

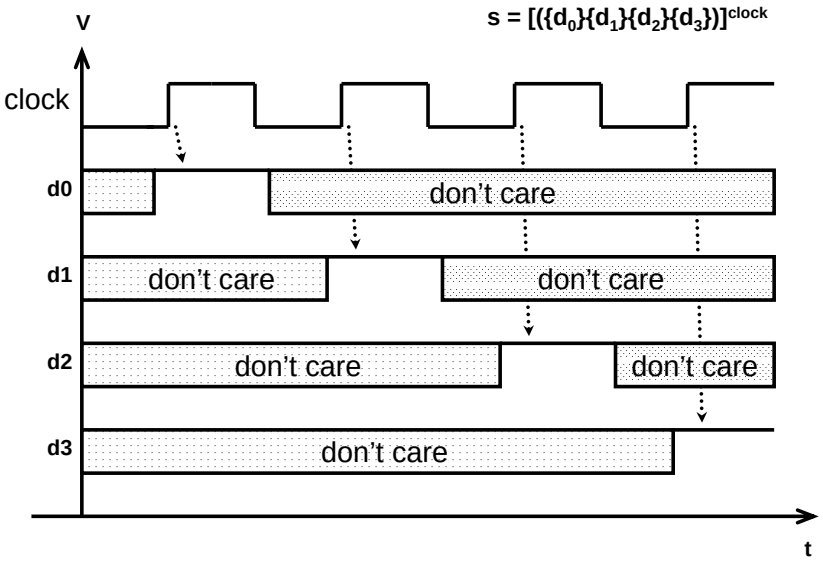


Figure 4.5: Example of voltage time line corresponding to C-sequence $s = [(\{d_0\}\{d_1\}\{d_2\}\{d_3\})]^{clock}$

**Definition 4.7** Given a set $C \subseteq AllC$, define the CLOCKING OPERATION $[\,]^c$ over a condition sequence $s = \{c_1, c_2 ... c_n\} \in \mathcal{L}$ such that $[s]^c$ represent the clocked sequence where condition set $C_i$ $(1 \leq i \leq n)$ is true during the $i^{th}$ pulse of clock condition set C. The set C is referred to as the clocking condition set.

In our case, we will have $\{c_{clk}\} \subseteq AllC$ as our synchronizing condition signal.

**Example 4.2** To illustrate the clocking ordering, consider a scanner model unit which output data signals to a PC. The data signals is represented as a clocked C-sequence, $s = [(\{d_0\}\{d_1\}\{d_2\}\{d_3\})]^{clock}$. Condition signal clock will be responsible to clock the data signals. Such clocked C-sequence would mean that the data signal $\{d_0\}$ will be true during the first rising edge of condition signal clock, the data value of condition signal $\{d_1\}$ will be true in the second rising edge of signal clock and so forth. Detailed description of the signal time line is shown in Figure 4.5.

As stated before, the condition model structure of inputblock as shown in Figure 4.1 is meant for providing general perspective on inputblock and also for describing the role for $C_{legal,IB}$ which are mainly legality identification. For C-sequence with multiple elements such as a clocked sequential signals, it will require condition system models as shown in Figure 4.6. The construction procedures for condition

models is presented in Algorithm 4.1. The approach of Algorithm 4.1 basically involves three major steps:

1. Given a clocked C-Sequence $s_{legal,IB} = [(\{C_{legal,1}\}\{C_{legal,2}\}.....\{C_{legal,N}\})]^{c_{clk}}$ where N represent the total number of legal condition sets in the given clocked C-sequence, algorithm 4.1 begins by first creating N number of nets $G_i$ as shown in Figure 4.7.

2. Next, we will connect nets $G_i$ together by creating arcs from $t_{i,1}$ of each net $G_i$ to each newly created place $p_{i,5}$. Transition $t_{i,1}$ is associated with condition $c_{clk}$. New arcs will then connect each place $p_{i,5}$ to a newly created transition $t_{i,4}$ which is associated with condition $\neg c_{clk}$. From the transition we will create an arc to place $p_{i+1,1}$. Place $p_{i,3}$ of each net $G_i$ will be assigned to set of places, $P_{legal,clk}$. Note that the value of $i$ will increase in one increment till it reaches the value of N.

3. Finally we will join the set of places $P_{legal,clk}$ to a transition $t_{join}$ and from $t_{join}$ to place $p_{legal,IB}$.

Figure 4.8 describes the condition model structure for identifying non-clocked C-sequence signal. Such condition model is in fact a special case for condition model shown in Figure 4.5. We will present the construction procedures for identifying non-clocked signal in Algorithm 4.2. Given a clocked C-Sequence $s_{legal,IB} = (\{C_{legal,1}\}\{C_{legal,2}\}.....\{C_{legal,N}\})$, we will create transitions $t_{legal}$ and associate each of them with a single unique legal condition set in the C-sequence. The transition is created in sequential order and the last legal transition $t_{legal}$ will connect to the place $p_{legal,IB}$.

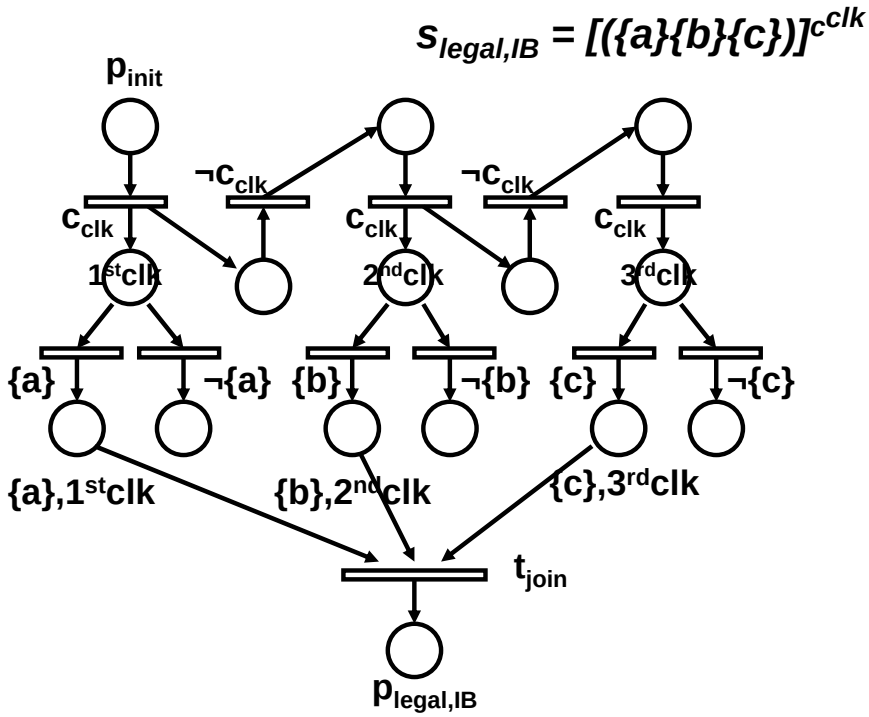


Figure 4.6: Condition system model of inputblock for clocked signal

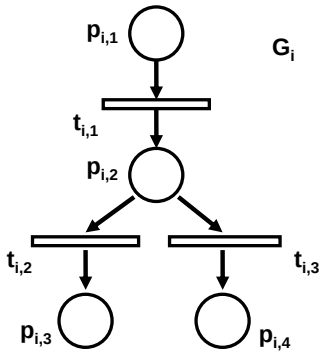


Figure 4.7: Figure for net $G_i$

Figure 4.8: **Algorithm 4.1** An algorithm for construction of inputblock for clocked signal.

1 Given a C-Sequence $s_{legal,IB}$=$[(\{C_{legal,1}\}\{C_{legal,2}\}\{C_{legal,3}\}...\{C_{legal,N}\})]^{c_{clk}}$.
2 **For** $1 \leq i \leq N$ {
3 Create net $G_i$ with structure as shown in figure 4.6.
4 $P_{legal,clk} \Leftarrow P_{legal,clk} \cup \{p_{i,3}\}$.
5 Define $\Phi(t_{i,1}) = \{c_{clk}\}$.
6 Define $\Phi(t_{i,2}) = C_{legal,i}$.
7 Define $\Phi(t_{i,3}) = \neg C_{legal,i}$.
8 }
9 **For** $1 \leq j \leq N-1$ {
10 Create transition $t_{j,4}$, define $\Phi(t_{j,4}) = \neg c_{clk}$.
11 Create place $p_{j,5}$.
12 Create arc from $t_{j,1}$ to $p_{j,5}$.
13 Create arc from $p_{j,5}$ to $t_{j,4}$.
14 Create arc from $t_{j,4}$ to $p_{j+1,1}$.
15 }
16 Relabel $p_{1,1}$ as $p_{init}$.
17 Create a place, $p_{legal,IB}$.
18 Create a transition $t_{join}$, define $\Phi(t_{join}) = \emptyset$.
19 Create arcs from each $p \in P_{legal,clk}$ to $t_{join}$.
20 Create an arc from $t_{join}$ to $p_{legal,IB}$.

$s_{legal,IB} = (\{a\}\{b\}\{c\})$

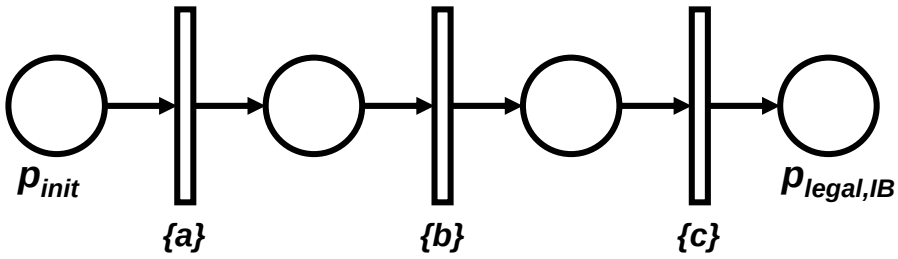


Figure 4.9: Condition system model of inputblock for non-clocked signal

Figure 4.10: **Algorithm 4.2** An algorithm for construction of inputblock for non-clocked signal.

1 Given a C-Sequence $s_{legal,IB}$=($\{C_{legal,1}\}\{C_{legal,2}\}\{C_{legal,3}\}...\{C_{legal,N}\}$).
2 Create place $p_1$.
3 Assign place $p_1$ as $p_{init}$.
4 **For** $1 \leq i \leq N$ {
5 Create transition $t_i$.
6 Define $\Phi(t_i) = C_{legal,i}$.
7 Create place $p_{i+1}$.
8 Create arc from $t_i$ to $p_{i+1}$.
9 }
10 **For** $1 \leq i \leq N-1$ {
11 Create arc from $p_i$ to $t_i$.
12 }
13 Assign $p_N$ as $p_{legal,IB}$.

### 4.5.2 Construction Procedures for Outputblocks

The task of the outputblock will be outputting the legal language from the auxiliary system to SUT (output command/data). The accomplishment of the task will allow the completion of signal interaction cycle between SUT and auxiliary system. In this section, we will present algorithm 4.3 and 4.4 to describe how we perform legality outputting by outputblock based on the clocking properties of outgoing C-sequence signals in similar approach as in inputblocks.

Algorithm 4.3 describes the construction procedure for output-block outputting clocked signal. Given output clocked C-Sequence $s_{legal,OB}=[(\{C_{legal,1}\}\{C_{legal,2}\}\{C_{legal,3}\}...\{C_{legal,N}\})]^{c_{clk}}$. The algorithm starts off by creating a place and connect an arc to a newly created transition where we will assign clock condition signal to it. From the transition we will create another a new place. We will assign an output condition set to this place. This process is repeated according to the number of legal condition sets in the C-sequence. Next we create another transition and assign negated clock condition signals to it. This transition is responsible for connecting the legal place to its next place.

In Algorithm 4.4 we will describe the construction procedure for outputblock outputting non-clocked parallel signal. Given output non-clocked C-Sequence $s_{legal,OB}=(\{C_{legal}\})$. we will create a place and connect arc to a transition. Completion condition sets of the Controller will be assigned to this transition. The arc connection from the transition to a place where legal output condition is assigned will mark the end of non-clocked outputblock construction.

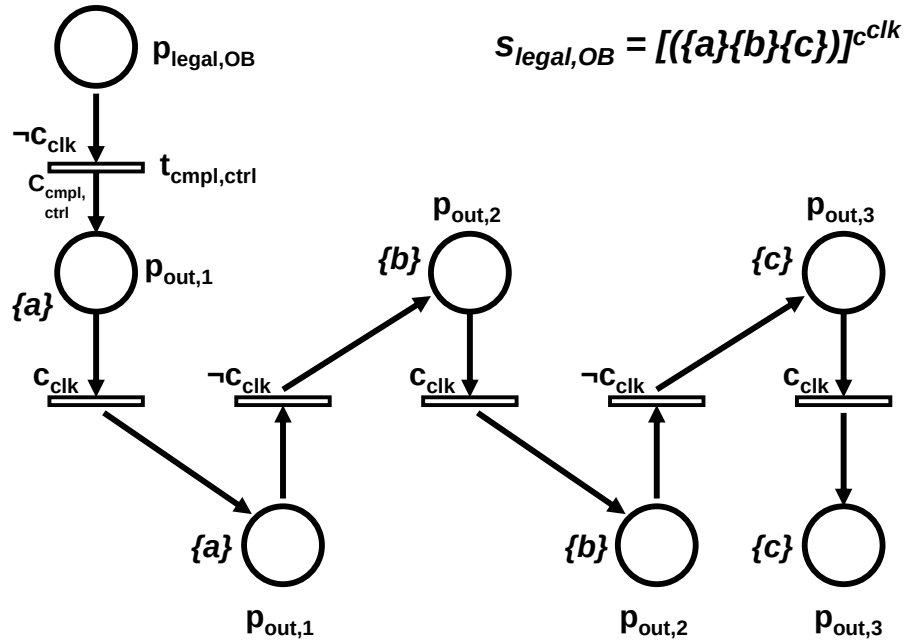


Figure 4.11: Condition system model of outputblock for clocked signals

Figure 4.12: **Algorithm 4.3** An algorithm for construction of outputblock for clocked signals.

1 Create place $p_{legal,OB}$.
2 Create transition $t_{cmpl,ctrl}$.
3 Define $\Phi(t_{cmpl,ctrl}) = \{\neg c_{clk}, c_{cmpl,ctrl}\}$.
4 Create arc from place $p_{legal,OB}$ to $t_{cmpl,ctrl}$.
5 Given a C-Sequence $s_{legal,OB}$=$[(\{C_{legal,1}\}\{C_{legal,2}\}\{C_{legal,3}\}...\{C_{legal,N}\})]^{c_{clk}}$.
6 Let N be the number of Condition Sets(C) in $s_{legal}$.
7 **For** $1 \leq i \leq N$ {
8 Create place $p_{i,1}$.
9 Create transition $t_{i,1}$.
10 Create place $p_{i,2}$.
11 Create arc from $p_{i,1}$ to $t_{i,1}$.
12 Create arc from $t_{i,1}$ to $p_{i,2}$.
13 Assign $p_{i,1}$ and $p_{i,2}$ as $p_{out,i}$.
14 Define $\Phi(p_{out,i}) = C_{legal,i}$.
15 Define $\Phi(t_{i,1}) = \{c_{clk}\}$.
16 }
17 **For** $1 \leq j \leq N-1$ {
18 Create transition $t_{j,2}$, define $\Phi(t_{j,2}) = \{c_{\neg clk}\}$.
19 Create arcs from $p_{j,2}$ to $t_{j,2}$.
20 Create arc from $t_{j,2}$ to $p_{j+1,1}$.
21 }
22 Create arc from $t_{cmpl,ctrl}$ to $p_{1,1}$.

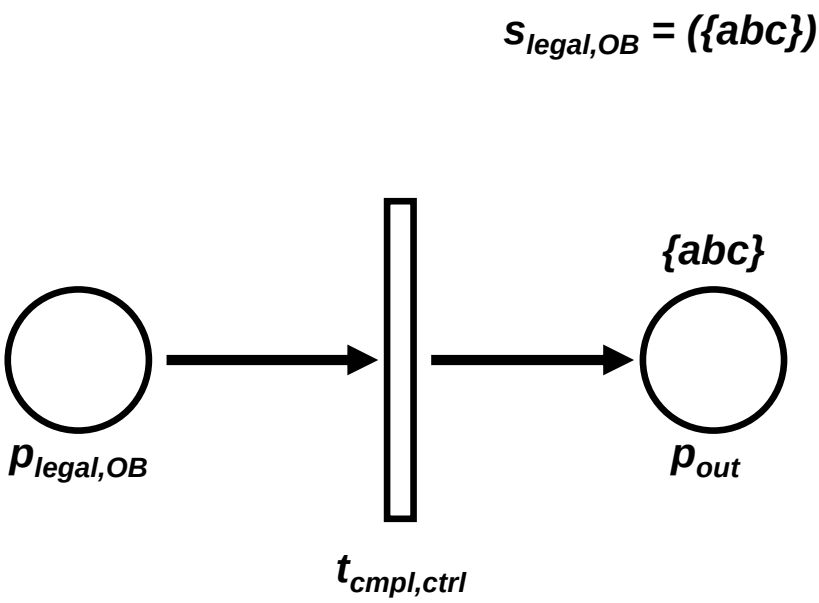


Figure 4.13: Condition system model of outputblock for non-clocked signals

Figure 4.14: **Algorithm 4.4** An algorithm for construction of outputblock for non-clocked signals.

1 Given a C-Sequence $s_{legal,OB}$=($\{C_{legal}\}$).
2 Create a place $p_{legal,OB}$.
3 Create a place, $p_{out}$.
4 Create a transition $t_{cmpl,ctrl}$.
5 Define $\Phi(t_{cmpl,ctrl}) = \{c_{cmpl,ctrl}\}$.
6 Create an arc from $p_{legal,OB}$ to $t_{cmpl,ctrl}$.
7 Create an arc from $t_{cmpl,ctrl}$ to $p_{out}$.
8 Define $\Phi(p_{out}) = C_{legal}$.

### 4.5.3 Construction Procedures for Virtualblocks

Under our modeling scheme, there are generally two main tasks for virtualblock to accomplish with respect to I/O interactions with SUT. The first task of the virtualblock is to recognize the incoming legal language from SUT to auxiliary system i.e. an input command or data. Upon recognizing the legal language from SUT, the second task of the virtualblock is to excite and output appropriate legal language from auxiliary system to SUT i.e. an output command or data. Note that the I/O signals from SUT can be modeled by a specblock. All of these functionality had been covered in previous sections.

In this section, we will describe the construction procedure for forming a virtualblock by connecting multiple inputblocks and outputblocks in algorithm 4.5 and also the activation of auxiliary system by the Controller in algorithm 4.6. In Algorithm 4.5, we will connect each place $p_{legal,IB}$ of inputblock to the corresponding place $p_{legal,OB}$ of outputblock.

In algorithm 4.6, place $p_{idle}$ represent a place marked under initial marking $m_0$ of the auxiliary system model and has emptyset $\emptyset$ as condition output. The algorithm will create the arc from place $p_{idle}$ to a transition which is assigned with the Controller's activation condition signal. Arcs are then created from the transition to place $p_{init}$ of each inputblocks previously created.

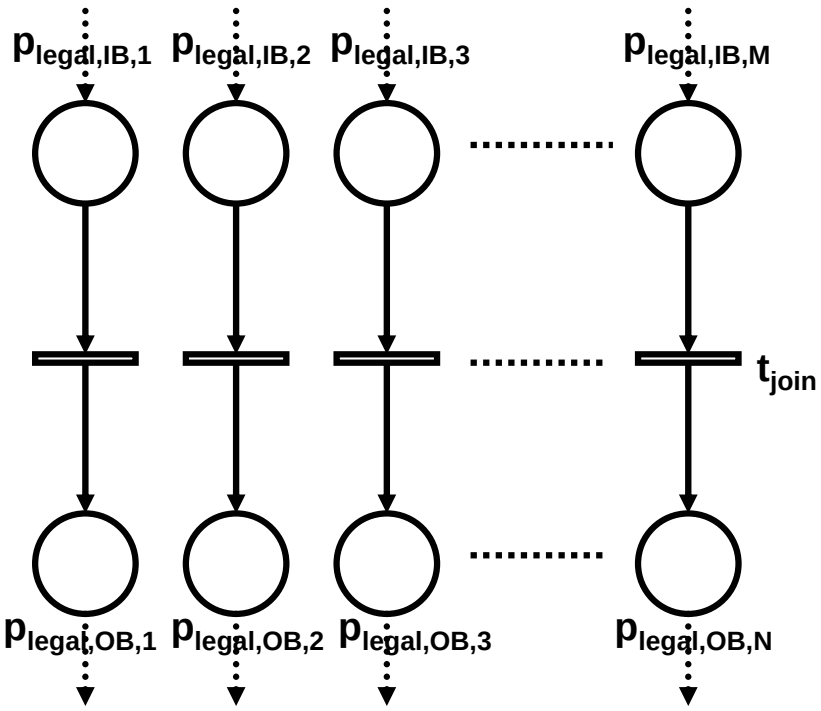


Figure 4.15: Condition system model of multiple inputblocks and outputblocks composition

Figure 4.16: **Algorithm 4.5** An algorithm for construction of multiple inputblocks and outputblocks composition.

1 Given $P_{legal,IB} = (p_{legal,IB,1}, p_{legal,IB,2}.....p_{legal,IB,M})$.
2 Given $P_{legal,OB} = (p_{legal,OB,1}, p_{legal,OB,2}.....p_{legal,OB,M})$.
3 Let M be the total number of places in each $P_{legal,IB}$ & $P_{legal,OB}$.
4 **For** i = 1 to M {
5 Create a transition $t_{join}$.
6 Define $\Phi(t_{join}) = \{\emptyset\}$.
7 Create arc from $p_{legal,IB,i}$ to $t_{join}$.
8 Create arc from $t_{join}$ to $p_{legal,OB,i}$.
9 }

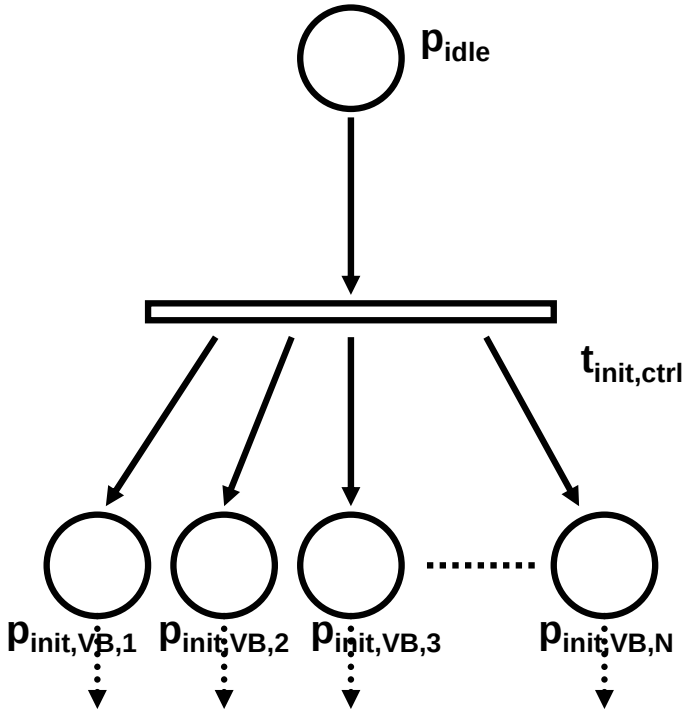


Figure 4.17: Condition system model of multiple virtualblocks initiation

Figure 4.18: **Algorithm 4.6** An algorithm for construction of multiple virtualblocks initiation.

1 Given set of places $P_{init,VB} = (p_{init,VB,1}, p_{init,VB,2}, p_{init,VB,3} ... p_{init,VB,N})$.
2 Let N be the total number of virtualblocks in auxiliary system.
3 Create a place $p_{idle}$.
4 Create a transition $t_{init,ctrl}$.
5 Define $\Phi(t_{init,ctrl}) = \{c_{init,ctrl}\}$.
6 Create an arc from $p_{idle}$ to $t_{init,ctrl}$.
7 Create arcs from $t_{init,ctrl}$ to $P_{init,VB}$.

# Chapter 5

# Fault Detection of Virtualblock

In this chapter, we present another building block of our modeling for interfacing framework: DETECTBLOCKS which is synthesized from virtualblock. The role of detectblock is to perform fault detection on target system which interacts with the system model. Before we introduce and describe the detectblock, we will first discuss about the relationship between our formal definition of fault and the detectblock in section 5.1. We then present the notion of DETECTABILITY. Next in section 5.3 we will present the formal definition of detectblock and also its general condition net model. We conclude this chapter by introducing the construction procedures for detectblock to perform fault detection and also the reset operation on both detectblock and virtualblock.

## 5.1 Fault

In chapter three, we formally define fault as a discrepancy between a real behavior of a system and an expected behavior of a system. The formal definition of fault would imply that if the incoming signal from target system is inconsistent with the expected system behavior of the system model then we will declare the incoming signal as a fault.

This definition of fault in general is sufficient to describe the role of the detectblock and the type of faulty behavior it is capable of detecting. However, the fault that is detected by the detectblock is slightly different than the fault in the formal definition due to the differences between expected language and legal language. Note that our system model $\mathcal{G}^{Aux,E,I}$ is the subsystem of expected auxiliary system model, $\mathcal{G}^{Aux,E}$. If the incoming signal is within the expected behavior of system model but beyond the system behavior of subsystem model then it will be detected by the detectblock as a fault. Briefly, the fault that is detected by detectblock can

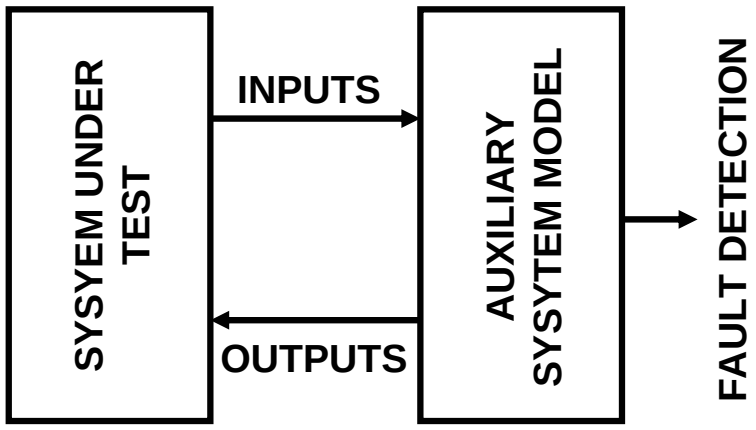


Figure 5.1: Scheme for Fault Detection of System Under Test and Auxiliary System

be described as a discrepancy between a real behavior of a system and a legal behavior of a system. In fact, the fault that detectblock is detecting evolved from our definition of fault in chapter 3.

Under our modeling framework, detectblock's fault detection operates on an observed C-sequence, $s$ from system under test (SUT) and when the observed C-sequence from SUT does not match any of the expected legal behaviors within the auxiliary system model interfacing SUT, $\mathcal{G}^{\mathrm{Aux,E,I}}$ then we will declare that a fault has occurred. In other words, the specblock of SUT $\mathcal{G}^{\mathrm{SUT,SB}}$ which is achievable within the expected auxiliary system $\mathcal{G}^{\mathrm{Aux,E}}$, may not always be achievable within its subsystem, the expected auxiliary system interfacing SUT $\mathcal{G}^{\mathrm{Aux,E,I}}$.

## 5.2 Detectability

Next, we will introduce the definition of DETECTABILITY over condition signals. Let $c_D \in AllC$ be a condition signal which is true whenever a fault is detected. Under our fault detection scheme, a system under test (SUT) is said to be DETECTABLE if its I/O interactions is within the auxiliary system and the signal interactions between SUT and auxiliary system contained information which is rich enough to describe the system functionality with respect to auxiliary system (i.e. a command or instruction to $G^{Aux}$). To further explain these, from the system information(specification) of auxiliary system, we will have the knowledge of expected input signals from the

actual target system (SUT) and also the expected output signals from auxiliary system to SUT. If these responses between SUT and auxiliary system are inconsistence within the expected legal behaviors then we would declare that fault has occurred and we said that the faulty system behavior in SUT is detectable. Note that if not all of the I/O interactions of SUT is within the auxiliary system then we said the SUT is PARTIALLY DETECTABLE by auxiliary system.

In order for our fault detection scheme to successfully detect the faults classified above, the following definition must be satisfied:

**Definition 5.1** Given real and expected systems $\mathcal{G}^{SUT,R}$, $\mathcal{G}^{SUT,E}$ and given auxiliary system $\mathcal{G}^{Aux,E,D}$ then $\mathcal{G}^{Aux,E,D}$ has detector capability for ($\mathcal{G}^{SUT,R}$, $\mathcal{G}^{SUT,E}$) if there exists a condition $C_D$ output from $\mathcal{G}^{Aux,E,D}$, such that:

1. For any $s \in L(\mathcal{G}^{SUT,R} \cup \mathcal{G}^{Aux,E,D})$, if $(\{\emptyset\}\{c_D\}) \leqslant s$ then $s \notin L(\mathcal{G}^{SUT,E})$.
2. For any $s \in L(\mathcal{G}^{SUT,R} \cup \mathcal{G}^{Aux,E,D}) \mid_{AllC-c_D}$ then $s \in L(\mathcal{G}^{SUT,R} \cup \mathcal{G}^{Aux,E})$.
3. For any $s \in L(\mathcal{G}^{D})$ and $s' \in L(\mathcal{G}^{Env})$, $(s \mid_{C_{in}(\mathcal{G}^{D})} \bigcap s' \mid_{C_{out}(\mathcal{G}^{Env})}) = \emptyset$.

Statement 1 simply states that if the concatenation of s include $c_D$ when it initiates from the initial condition then this would indicate that there is a fault occurred in that particular C-sequence, s. Statement 2 states that despite the detector capability of $\mathcal{G}^{Aux,E,D}$, $\mathcal{G}^{Aux,E}$ and $\mathcal{G}^{Aux,E,D}$ are essentially the same. And finally statement 3 states that I/O signals within system with detector capability $\mathcal{G}^{Aux,E,D}$ and the external environment $\mathcal{G}^{Env}$ are mutually exclusive. The last statement makes sure that the system with detector capability is not subjected to any noise or random signal from the external environment.

## 5.3 Detectblock

In this section, we will define the system properties of detectblock, a system with detection capability. The general structure of condition system model for detectblock is shown in Figure 5.2. The box in Figure 5.2 similar to inputblock's in Figure 4.1 contains different kinds of condition model nets and their structure will depend upon types of condition signals the detectblock is detecting (clocked/non-clocked). Despite the difference, every box will at least consist of a transition denoted as $t_{cmpl,ctrl}$. We will formally define detectblock in the following definition:

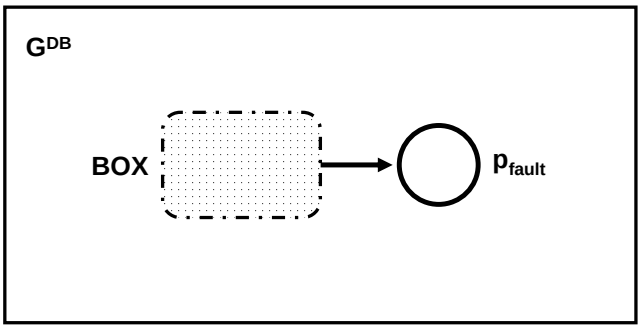


Figure 5.2: General Structure of Detectblock

**Definition 5.2** Detectblock is a form of condition system model defined as a doublet $(\mathcal{G}^{\mathrm{DB}}, \mathfrak{m}_{\mathrm{fault}})$, where $\mathcal{G}^{\mathrm{DB}}$ is a condition system and $\mathfrak{m}_{\mathrm{fault}} \in M_{\mathcal{G}^{\mathrm{DB}}}$ is a FAULT STATE. Detectblock can be characterized by a condition sets, defined as: $C_{\mathrm{cmpl,ctrl}} = \Phi_{\mathcal{G}^{\mathrm{DB}}}(t_{\mathrm{cmpl,ctrl}})$.

Similar to outputblock's, the condition set $C_{\mathrm{cmpl,ctrl}}$, represents a set of conditions that is true whenever the Controller issues the completion signal to system model. In our cases, $C_{\mathrm{cmpl,ctrl}}$ will be a singleton condition set. Whenever output place of transition $t_{\mathrm{cmpl,ctrl}}$, $p_{\mathrm{fault}}$ is marked then it would indicate that a fault had occurred.

## 5.4 Algorithm

The fault detector of auxiliary system model performs fault detection based on signals originated from system under test (SUT). The ultimate task of the detector under our thesis issues will be to perform fault detection on incoming signals from SUT based on pre-defined specification of auxiliary system. We will discuss about these tasks in detail in the following subsection.

### 5.4.1 Construction Procedures for Detectblocks

After we have the capability to recognize the legal language from system under test, our task would be to detect fault from the system under test. The fault detection is convenient in the sense that since we categorized fault as anything that is dif-

ferent from the expected legal response. As long as the signals from SUT is not within the expected legal language then the fault detection signal will be triggered. Detectblock is synthesized from virtualblock or more precisely the inputblock since inputblock's task is to recognize the incoming signals from SUT.

Algorithm 5.1 & 5.2 will describe how we perform fault detection on non-clocked & clocked signals of inputblock. In algorithm 5.1 we perform fault detection on non-clocked signal by creating transition $t_{cmpl,ctrl}$ which is associated with the condition signal $c_{cmpl,ctrl}$. The number of transition $t_{cmpl,ctrl}$ created is depended on the number of transition $t_{legal}$ found in a particular inputblock. We will connect arc from input place of transition $t_{legal}$ which is associated with condition set $C_{legal}$ where $C_{legal} \neq \emptyset$ to a transition $t_{cmpl,ctrl}$ on 1 to 1 basis. From all transitions $t_{cmpl,ctrl}$ created, arcs are connected to a single newly created place $p_{fault}$. This completed the procedures for algorithm 5.1.

In algorithm 5.2, we will perform fault detection on inputblock of clocked signal. First of all we begin by determining the number of places in set of place $P_{legal,clk}$ of an inputblock. The total number of places found is denoted as N. We then create a place $p_{fault}$. Next we create a transition $t_{N,4}$ with negated clocked signal $\neg c_{clk}$ associate with it. This transition is connected by its input place which is a newly created place $p_{N,5}$. $p_{N,5}$ is also the output place of the transition $t_{N,1}$ of inputblock. A newly created place $p_{N+1,1}$ will serve as the output place of transition $t_{N,4}$. A transition $t_{N+1,1}$ with clocked signal $c_{clk}$ associate with it is then created. An arc is created from $p_{N+1,1}$ to $t_{N+1,1}$ and from the transition $t_{N+1,1}$ arcs will be created to connect to set of places $P_{clk} = (p_{clk,1}.....p_{clk,N})$. Each of the place of $P_{clk}$ and $P_{legal,clk}$ will serve as the input place to each transition $t_{cmpl,ctrl}$ and the place $p_{fault}$ will serve as the output place of the transition.

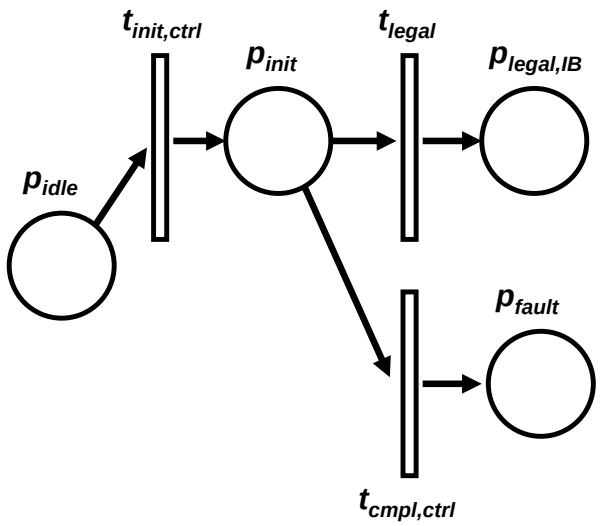


Figure 5.3: Condition system model for non-clocked inputblock fault detection

Figure 5.4: **Algorithm 5.1** An algorithm for non-clocked inputblock fault detection.

1 Given inputblock of non-clocked signals.
2 Create a place, $p_{fault}$.
3 **For** all transition t s.t. $\Phi(t) \neq \emptyset$ and $\Phi(t) = C_{legal}$ {
4 Create a transition $t_{cmpl,ctrl}$.
5 Define $\Phi(t_{cmpl,ctrl}) = \{c_{cmpl,ctrl}\}$.
6 Create arc from the input place of t to $t_{cmpl,ctrl}$.
7 Create arc from $t_{cmpl,ctrl}$ to $p_{fault}$.
8 }

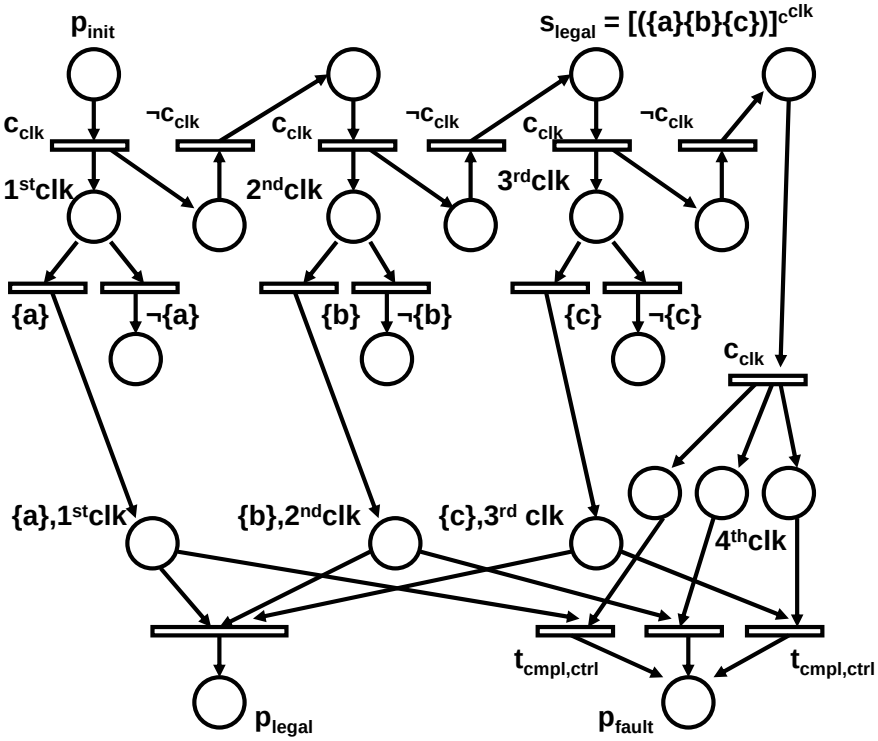


Figure 5.5: Condition system model for clocked inputblock fault detection

Figure 5.6: **Algorithm 5.2** An algorithm for clocked inputblock fault detection.

1 Given inputblock of clocked signals.
2 Let N be the total number of places in set of places $P_{legal,clk}$ of inputblock.
3 Create a place, $p_{fault}$.
4 Create transition $t_{N,4}$.
5 Define $\Phi(t_{N,4}) = \{c_{\neg clk}\}$.
6 Create place $p_{N,5}$.
7 Create arc from $t_{N,1}$ to $p_{N,5}$.
8 Create arc from $p_{N,5}$ to $t_{N,4}$.
9 Create arc from $t_{N,4}$ to $p_{N+1,1}$.
10 Create transition $t_{N+1,1}$.
11 Define $\Phi(t_{N+1,1}) = \{c_{clk}\}$.
12 Create arc from $p_{N+1,1}$ to $t_{N+1,1}$.
13 Create set of places $P_{clk} = (p_{clk,1}.....p_{clk,N})$.
14 Create arcs from $t_{N+1,1}$ to $P_{clk}$.
15 **For** j = 1 to N {
16 Create transition $t_{cmpl,ctrl}$.
17 Define $\Phi(t_{cmpl,ctrl}) = \{c_{cmpl,ctrl}\}$.
18 Create arc from $p_{legal,clk,j}$ to $t_{cmpl,ctrl}$.
19 Create arc from $p_{clk,j}$ to $t_{cmpl,ctrl}$.
20 Create arc from $t_{cmpl,ctrl}$ to $p_{fault}$.
21 }

### 5.4.2 Construction Procedures for Composing Multiple Detectblocks

In this subsection, we will present construction procedures for composing multiple detectblocks constructed in algorithm 5.1 & 5.2 to perform fault detection on the entire command cycle from SUT to auxiliary system. For each detectblocks constructed within an inputblock, we will have a place $p_{fault}$ to indicate whether there is a fault exist in the corresponding block. This however is not sufficient to indicate the functionality of an entire virtualblock. Note that under our fault detection methodologies, a virtualblock is considered in working condition as long as one of its inputblocks is not in fault state at the end of each command cycle. Therefore in our cases fault detection in a single detectblock is not adequate and we need a procedure to compose all the detectblocks and perform the overall system fault detection.

Such procedure will be presented in algorithm 5.3. We will define $p_{totalfault}$ as a place with fault detection condition signal $c_D$ corresponding to the entire command cycle from SUT. In algorithm 5.3, we will create a transition $t_{d,ctrl}$ and assign the condition signal $\{c_{d,ctrl}\}$ to it. Arcs will be connected from places $p_{fault}$ found in detectblock to this transition. From the transition, an arc will be connected to the place $p_{totalfault}$. At the end of each command cycle, the controller will issue the fault detection signal $\{c_{d,ctrl}\}$ to detectblocks. If any of the $p_{fault}$ of detectblocks is marked then the transition $t_{d,ctrl}$ will fire and transfer the corresponding token to place $p_{totalfault}$.

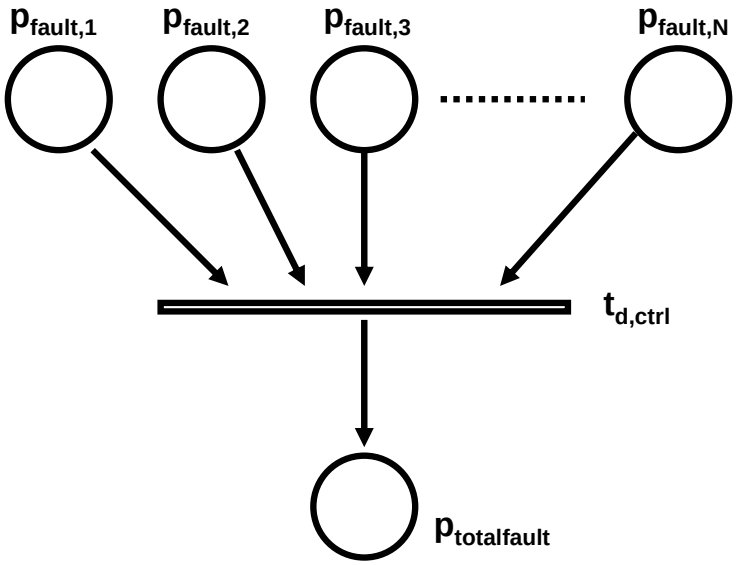


Figure 5.7: Condition system model for multiple detectblocks fault detection

Figure 5.8: **Algorithm 5.3** An algorithm for multiple virtualblock fault detection.

1 Given multiple detectblocks with set of places $P_{fault} = (p_{fault,1}.....p_{fault,N})$.
2 Let N be the total number of places in $P_{fault}$.
3 Create a transition $t_{d,ctrl}$.
4 Define $\Phi(t_{d,ctrl}) = \{c_{d,ctrl}\}$.
5 Create a place, $p_{totalfault}$.
6 Define $\Phi(p_{totalfault}) = \{c_d\}$.
7 Create arcs from $P_{fault}$ to $t_{d,ctrl}$.
8 Create arc from $t_{d,ctrl}$ to $p_{totalfault}$.

### 5.4.3 Construction Procedures for Resetting Virtualblock and Detectblock

In this final section of Chapter 5, we will introduced the reset operation for virtualblock and detectblock. The reset operation of virtualblock and detectblock is essential for the operability of the entire auxiliary system model. At the end of each command cycle from SUT, there will be tokens left in the output places $p_{out}$ of outputblocks or the fault places $p_{fault}$/$p_{totalfault}$ of detectblocks. These tokens will affect the functionality of auxiliary system model if they are not removed from their original places to an appropriate place at the beginning of next command cycle. The main task of reset operation is to remove these redundant tokens from output places and fault places of outputblock and detectblock to a place denoted as $p_{dump}$ at the beginning of each command cycle. This operation will ensure the correct operation of virtualblock and detectblock for each initial signals cycle.

The construction procedure to incorporated reset operation in virtualblock and detectblock will be presented in algorithm 5.4. The algorithm is achieved by creating transitions $t_{init,ctrl}$ and connect an arc from each places $p_{out}$, $p_{fault}$ & $p_{totalfault}$ found in virtualblock and detectblock to each of their own transition $t_{init,ctrl}$. Another arcs will then be connected from these transitions to a common place $p_{dump}$. Upon the issuance of condition signal $c_{init,ctrl}$ from the controller, each transitions $t_{init,ctrl}$ of virtualblocks and detectblock will fire and transfer the appropriate token to place $p_{dump}$ if there is any token in places $p_{out}$, $p_{fault}$ or $p_{totalfault}$.

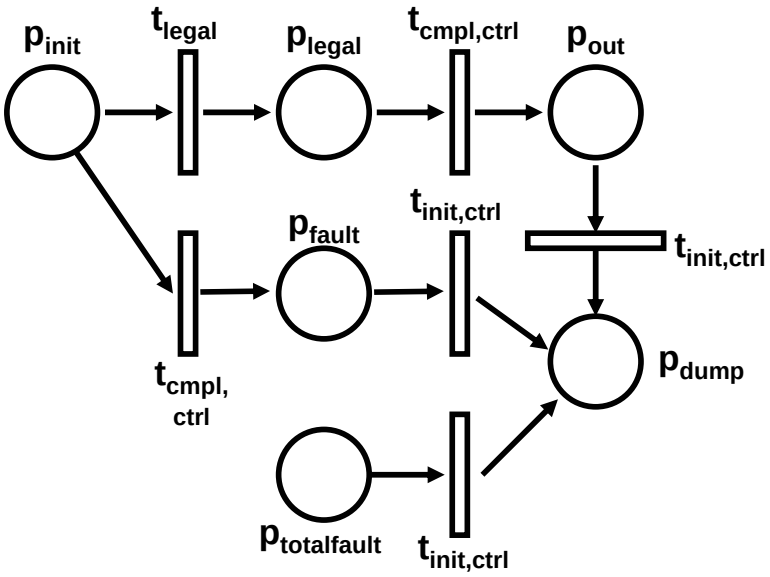


Figure 5.9: Condition system model of VirtualBlock with Reset Operation

Figure 5.10: **Algorithm 4.7** An algorithm for resetting virtualblock.

1 Given virtualblock and detectblock with places $p_{out}$, $p_{fault}$ and $p_{totalfault}$.
2 Let N be the total number of corresponding places.
3 Create a place, $p_{dump}$.
4 Create N number of transitions $t_{init,ctrl}$.
5 Define $\Phi(t_{init,ctrl}) = \{c_{init,ctrl}\}$.
6 Assign each place with an individual transition $t_{init,ctrl}$.
7 Create arc from each place to its own $t_{init,ctrl}$.
8 Create arcs from $t_{init,ctrl}$ to $p_{dump}$.

# Chapter 6

# Application to Scanner Control Unit

In this chapter, we present the application of our approaches by incorporating our modeling and fault detection methodologies into a scanner control unit of a printer. Our scanner control unit typically consists of two basic components: Charge Coupled Device(CCD) and Analog to Digital Convertor (ADC) which interact with the system under test (SPIN). SPIN issues input commands to both CCD and ADC, and also receives output from ADC. CCD is responsible for image capturing in the scanner. Generally CCD is a collection of light-sensitive photo diodes which convert photons (light) into electrons (electrical charge). In our application, we use Toshiba CCD linear image sensor, TCD 2558D as our CCD. And we will use ADC to process and digitise analogue output signals from TCD 2558D, the CCD sensors. Wolfson's 20 MSPS 16-bit CCD digitiser (WM8199) is used as our Analog to Digital Convertor. We also have a PC unit (the Controller) which interacts with SPIN internally (unobservable) and issues activation, completion signals and fault detection signals to CCD and ADC.

There are several factors favoring the design of systematic automated modeling and fault detection mechanisms for the scanner control unit systems: 1) Detecting failures in these systems is a complex task for a human monitoring the system since the tester has to respond to signals coming from various parts of the system. 2) Human's fault detection mechanism relies on expert system, and any changes in system design and system composition will complicate fault detection action since it would mean the total or partial reconfiguration of fault detecting process. 3) There are parts or components of system which are hard to access or observable to human tester, and therefore unable to perform fault detection on them.

With our system modeling mechanism, we will be able to resolve the issues encountered above. First of all, our mechanisms do not require modeling of the entire system which is complex and huge, but rather only the modeling of crucial

parts of the system which is necessary and sufficient for fault detection processes. This would greatly simplify the processes. Our system modeling mechanism also allow the reuse of model of system components and these resolve the issues of changes in design or system composition. Finally observability is not an issue in our case since we perform fault detection based on signal interaction from the system under test, and these signal interactions are observable to us.

Figure 6.1 illustrate the overview of scanner control unit system as described above. We illustrate our systematic approach to modeling and fault detection by considering two different types of signals found in this system: Binary Signal and Serial signal.

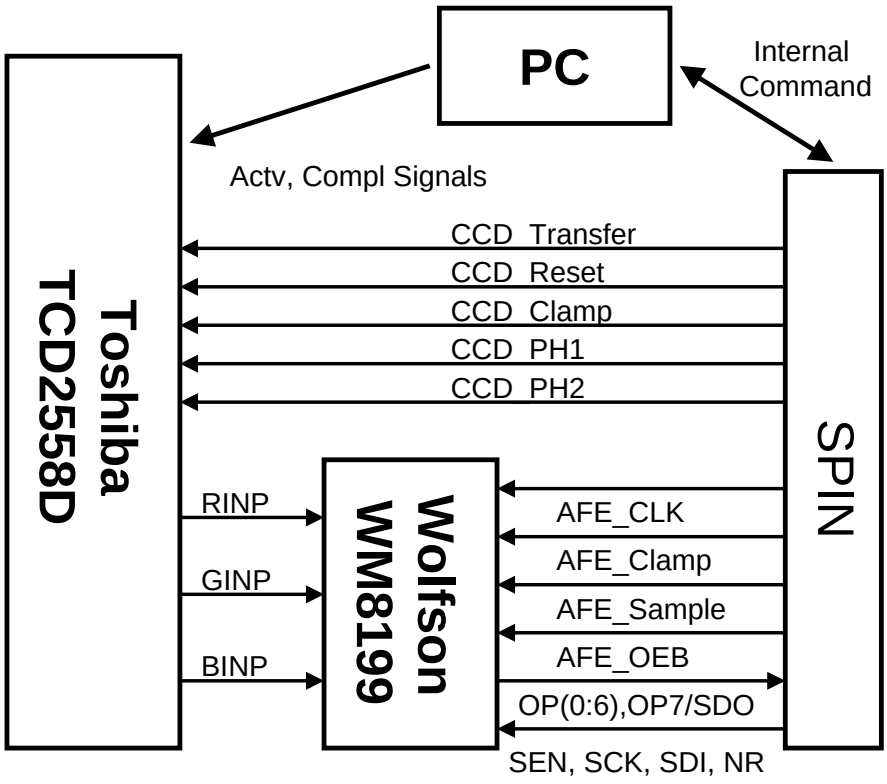


Figure 6.1: Figure for Scanner Control Unit Connections

## 6.1 Binary Signal

We will illustrate our systematic approach for modeling and fault detection of binary signal command in a system by first examining the specification of TCD 2558D. The specification of the CCD typically consist of a circuit diagram and timing chart. From the specification we will determine the modeling of the system by filtering out the redundant system. For example the photo diodes of the CCD are considered redundant system because we do not have complete information of the system and more importantly it doesn't constitute to the formation of legal command or language from the SPIN which is our system under test. Figure 6.2 illustrates one out of three similar parts of TCD2558D which generally consist of photo diodes, shift gate, analog shift register and clamp.

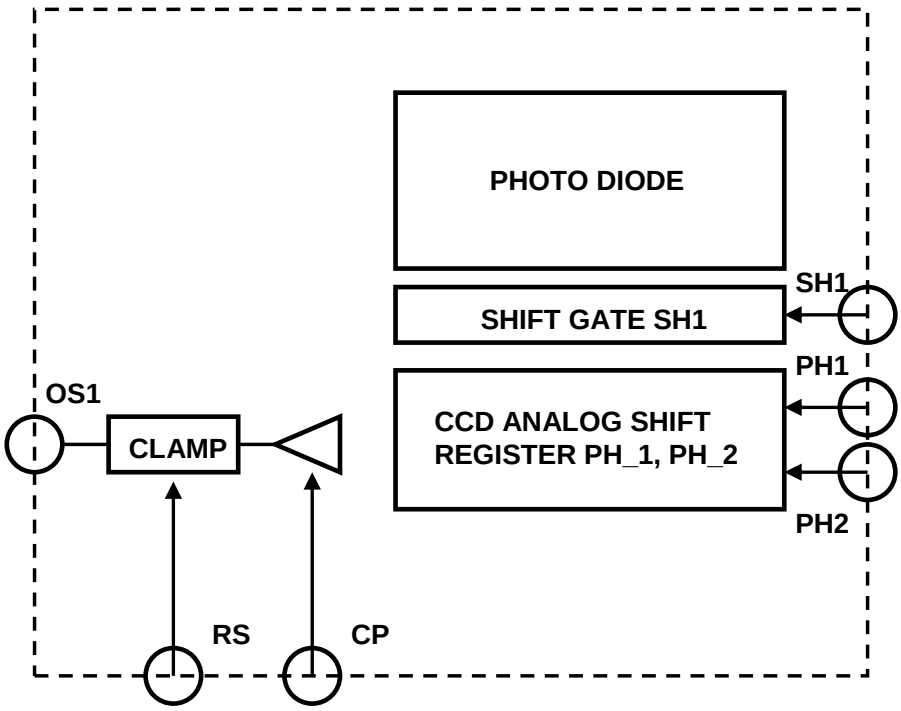


Figure 6.2: Figure for Part of TCD2558D

Next we will examine the timing chart. From the timing chart we can determine the legal input command from SPIN. Figure 6.3 illustrates the bit clamp mode of TCD 2558D which illustrate the pattern of signal SH, $\Phi 1$, $\Phi 2$, $\neg RS$, $\neg CP$ and its resulting signal OS. From this figure, we can determine that when SH, $\Phi 1$, $\neg RS$,

¬CP is high and Φ2 is low at the same time, then OS will go high from low which represent a fur

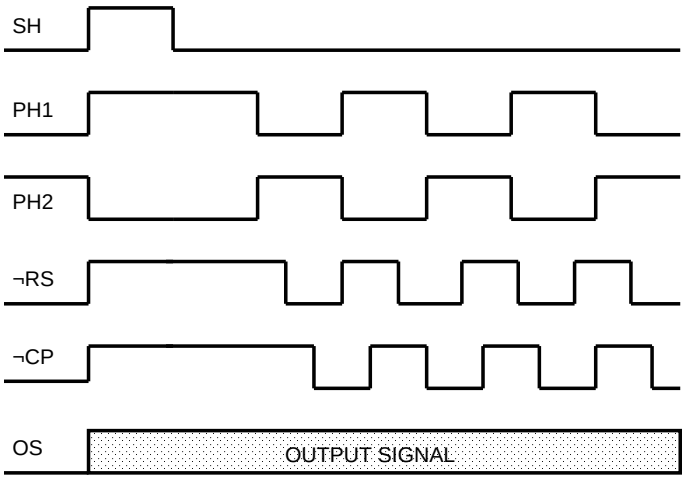


Figure 6.3: Timing Chart for TCD2558D in Bit Clamp Mode

Condition models in Figure 6.4 illustrate the modeling and fault detection mechanism model derived from the timing chart of TCD 2558D bit clamp mode. We first create a place, $p_{idle}$ with $m_0$ as our initial marking. Sequence of commands for bit clamp mode of TCD 2558D will be identified and assign to each empty transition. In our case, the first command for bit clamp mode will be the $PC_{init}$ signal from the PC which activate the CCD. The firing of $PC_{init}$ will reconfigure the marking and take the token from $p_{idle}$ to a new place. The second command will come as a form of Condition Sets which include five different signals and assign them into one single empty transition as input conditions. Next we will identify the outputs which result from the execution of these legal commands. A place is then to be created and be assigned with OS, the output signal as output condition. So when the legal transition (transition with legal command) fired, it will enabled the output place of OS. By the end of the command cycle, the PC will issue a signal, $PC_{cmpl}$ to indicate the completion of task. We will take advantage of this in our fault detection scheme by assigning the signal to an empty transition and let the place which is an input place of a legal transition as the input place to this transition too. In this way, we will be able to detect a fault since if the token is still in the input place of legal transition by the end of the cycle then this will mean that there is an error in the commands from SPIN. And by the time the signal $PC_{cmpl}$ is issued the token in the place will fire and lead it to the place, $p_{fault}$. Following that, another signal $PC_d$

will be issued to transfer the token from $\mathrm{p}_{\mathrm{fault}}$ to place $\mathrm{p}_{\mathrm{totalfault}}$ to indicate the completion of fault detection process.

We also need to indicate that for every end of cycle of commands, we will reset the auxiliary system model with $\mathrm{PC}_{\mathrm{init}}$ as a transition which transfer the token from the corresponding places to a place, called $\mathrm{p}_{\mathrm{dump}}$.

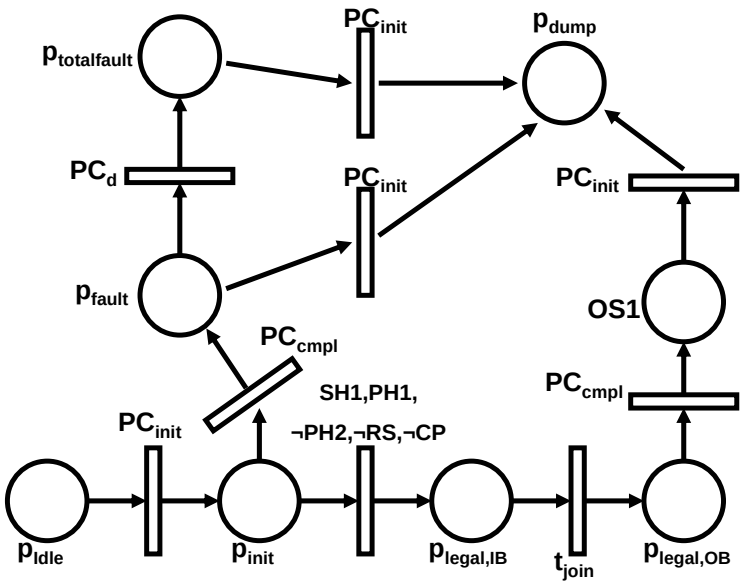


Figure 6.4: Condition Models for Binary Signal of TCD2558D in Bit Clamp Mode

## 6.2 Serial Signal

Next, we will illustrate our systematic approach for modeling and fault detection of serial signal in a system by first examining the specification of WM 8199. The specification of the ADC will also typically consist of a circuit diagram and timing chart. We will determine the modeling of the system by taking out the redundant system such as the RLC, CDS, Offset DAC, PGA and ADC of WM8199 because we do not have complete knowledge of the system and it doesn't constitute to the formation of legal command or language from the SPIN which is our system under test. Figure 6.5 illustrate one out of three similar part of WM8199 which generally consist of RLC, CDS, Offset DAC, PGA, ADC and Data I/O port. In our case we will only considered the components such as configurable serial control interface, timing control and Date I/O port.

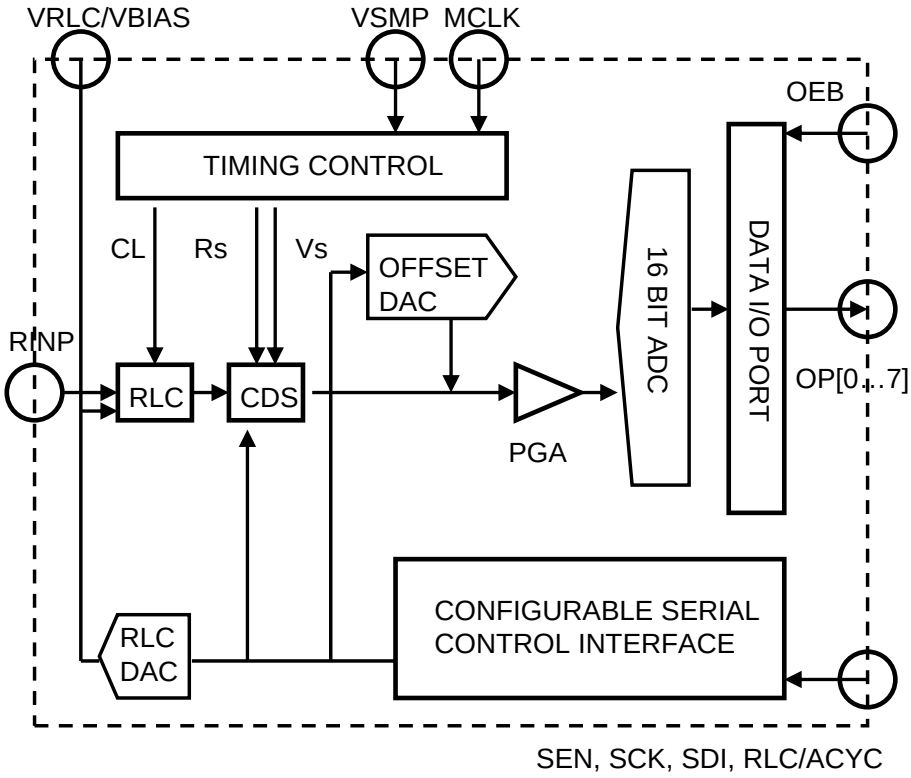


Figure 6.5: Figure for Part of WM8199

Next we will examine the timing chart. From the timing chart we can determine the legal input command from SPIN. Figure 6.6 illustrate the serial interface register write back mode for TCD2558D which illustrate the pattern of signal SCK, SDI, SEN, OEB and its resulting signal, SDO. Note that due to simplicity and limited space

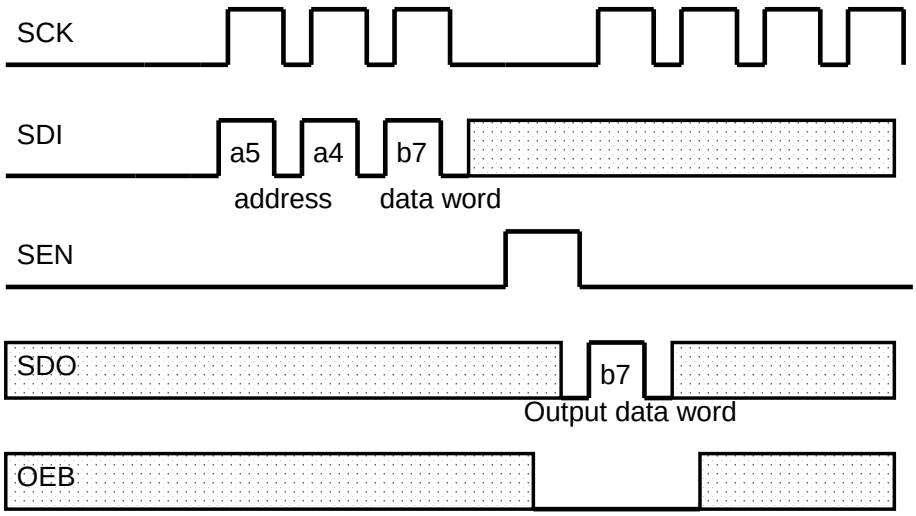


Figure 6.6: Timing Chart for WM8199 in Register Write Back Mode

issues, only address signals a5, a4 and part of data word signal, b7 is illustrate in this chart. From the chart, the read-back is initiated with address bits a5, a4 are set to 1, followed by data bit b7. When the data has been shifted into the device, a pulse is applied to SEN to transfer the data to SDO with OEB being held low.

Condition models in Figure 6.7 illustrate the fault detection mechanism model derived from the timing chart of WM 8199 register write back mode for address signal: $a5, a4$. As we know the configuration of these address signal will represent a legal command. If any unexpected composition or sequences of signals occur then these will indicate a fault. We compose our fault detection scheme by first identifying the places represent element of commands and theirs appropriate sequence. We then create an additional clock sequence from the original, for example in our case it will be the fourth clock. Analogues to the methodology of binary signal, every time $\mathrm{PC}_{\mathrm{cmpl}}$ of a transition is enabled it will assume that the input place is empty. If it is not, then fault will be detected by the firing of $\mathrm{PC}_{\mathrm{cmpl}}$ which transfer the token from the place to $\mathrm{p}_{\mathrm{fault}}$ and consequently to $\mathrm{p}_{\mathrm{totalfault}}$ through the firing of $\mathrm{PC}_{\mathrm{d}}$. Upon firing of $\mathrm{PC}_{\mathrm{cmpl}}$, the auxiliary system model will also output condition signal b7 through $\mathrm{p}_{\mathrm{out}}$.

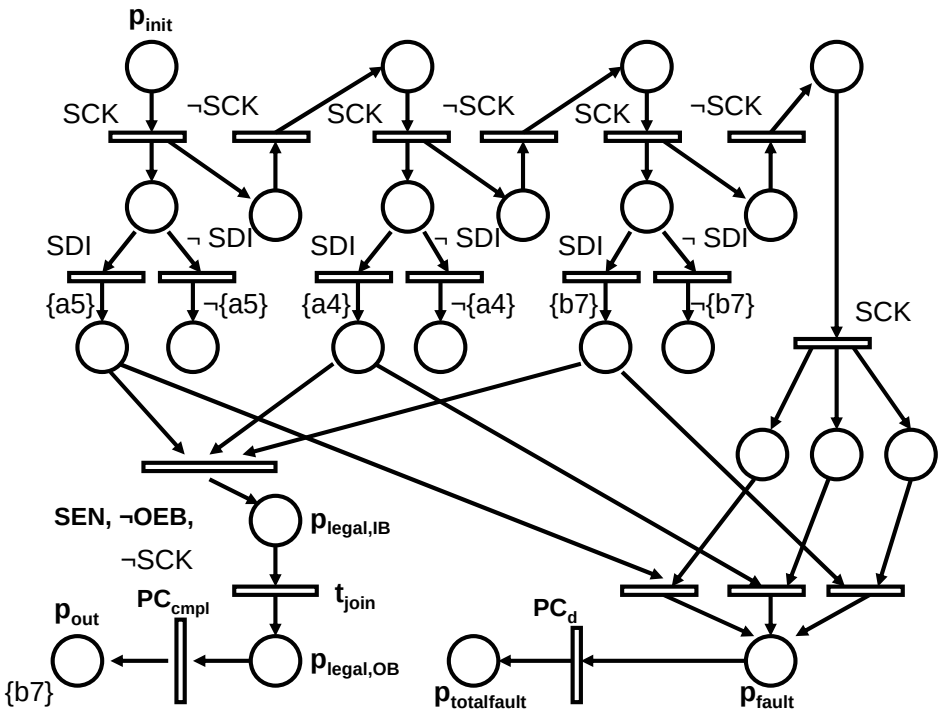


Figure 6.7: Condition Models for Serial Signal of Serial Interface: a5,a4

## 6.3 Software Overview

The modeling and fault detection methodologies presented in this thesis can be implemented by a software tool titled SPECTOOL first introduced in [HGSA00]. The software is written in Microsoft Visual C++ 6.0 and runs under Windows operating system.

In general, the goal of Spectool is to take a high-level description (the specnet) of the desired closed loop system behavior and develop a controller that will implement this behavior. The high-level description describes the sequential (and concurrent) desired inputs and outputs that the closed loop system should exhibit. A set of component models, describing the behaviors of elements of the system, are then analyzed in the context of the specnet. Modular controllers called taskblocks (Actionblocks and Maintainblocks) are then synthesized to drive these individual components to subgoals. These subgoals may be directly specified in the specnet, or may be indirectly specified through other taskblocks that require other components to be in specified states. Each taskblock is represented as a condition system model, the same representation as the specnet and the component nets. The logic of the taskblocks can be seen in the Spectool net editing software. The resulting controller is a collection of taskblocks that interact sequentially, concurrently, and hierarchically to control the system within the specified behavior.

A specnet can be used with component models to synthesize a controller, or can be used by itself as a way of specifying the control directly:

- For Control Synthesis: The conditions on the specnet places represent output conditions for the system. Thus, analysis of component models is necessary in order to determine a controller that drives the system through these outputs.
- For Control Specification: If the conditions on the specnet places represent output conditions for the controller (and thus input conditions for the system), then the specnet represents a direct specification of the control logic. In this case, conditions associated with transitions can be response conditions from the system.

The distinction between Control Synthesis and Control Specification depends on whether the output conditions describe the desired system outputs or the control outputs. For a given condition on a place, spectool tries to find a component net

that describes how the condition will be output from the system, and an actionblock is then synthesized for the condition. If no such component net is found, then spectool assumes that it must drive the signal directly. Thus, it is possible to create a specnet that contains both a specification of desired system behavior (for which control is synthesized), and some explicit specification of control outputs. Under our modeling methodologies, we will use the second approach Control Specification to implement our application.

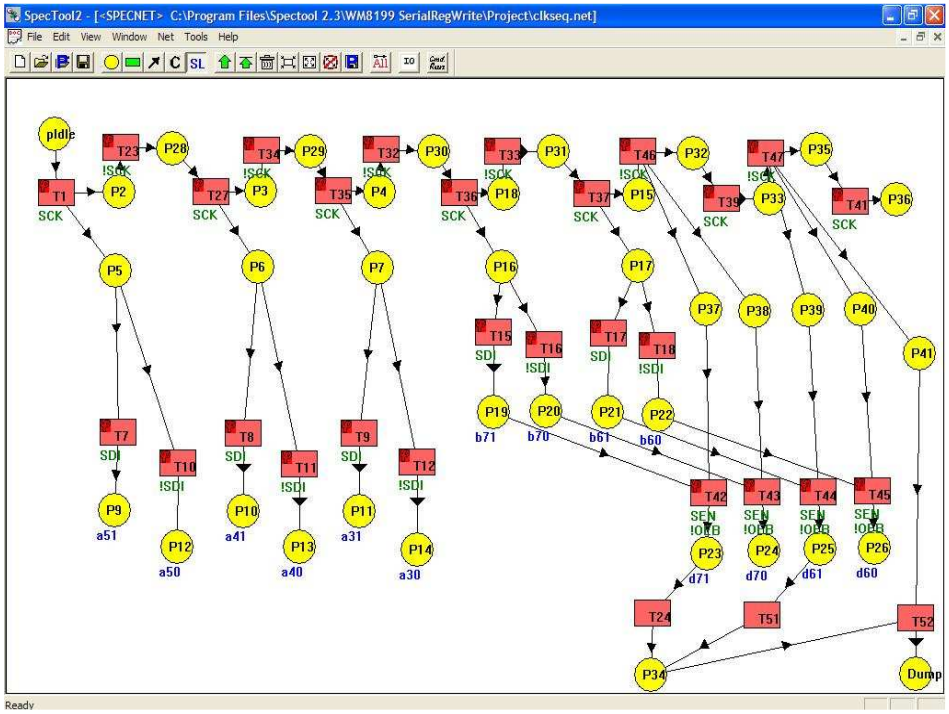


Figure 6.8: An example Specnet of scanner control unit in register write back mode

After the controller logic has been synthesized, it can be translated into C++ code. A transformed version of the specnet, called the Especnet, and all the synthesized taskblocks are passed to the software CodeMaker which generates C++ code for each. Finally, the resulting C++ files are compiled together into an executable, "generatedcode.exe.exe." This executable then interfaces through a shared memory to either a simulator (e.g. IODriverDummy.exe) or to an I/O hardware driver (e.g. IODriver.exe). The interface is standardized so that the synthesized controller is independent of the specific hardware used, so that a variety of different hardware drivers can be written independently of the Spectool project. In our particular case, the interface is a common commercial digital I/O board (Data Translation DT2820)

within the computer.

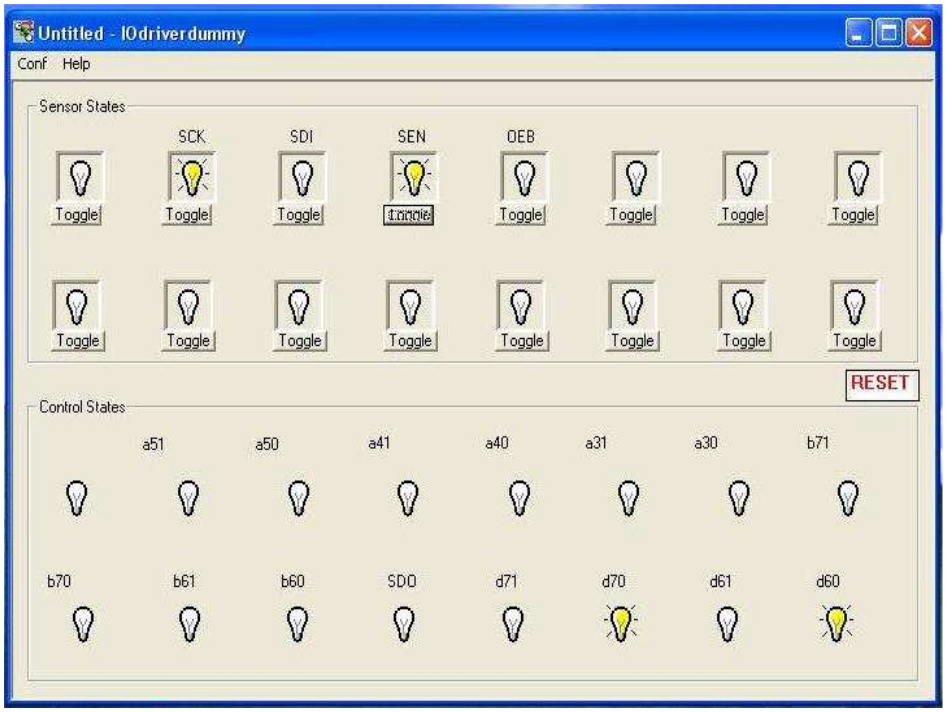


Figure 6.9: An example simulator of scanner control unit in register write back mode

# Chapter 7

# Conclusion

In this thesis, we have considered the problem of modeling and fault detection for complex systems from the perspective of discrete event systems. Our modeling methodology relies on modeling of system supplementary to system under test in the condition system Petri net framework and on the issues of fault detection we focus only on model-based offline passive fault detection. Our methodology does not rely on any test inputs to detect system failure. From our point of view, a fault, failure or any other faulty terms will be treated the same. Fault is defined as an inconsistency between observed system behavior from system under test with the expected system behavior of auxiliary system. Fault detection will be defined as the determination that the system under test is not behaving as expected according to the model of the auxiliary system.

In most model-based fault detection schemes however, they perform fault detection based on the modeling of system under test. Their fault detection schemes operate by obtaining the description of system under test in its normal and faulty modes. The issues in such fault detection schemes include modeling accuracy and the complexity in faulty system modeling. Therefore the success of such fault detection scheme will not only be relying on one capability to understand and obtain the complete if not sufficient information about the dynamic of system under test, but also the ability to model faulty systems which often are complex and exponentially large in size. One other issue of such fault detection scheme is system observability. There are subsystems in a system under test which is hard to access and therefore unobservable in some cases. If the corresponding sensor system fail to operate on such subsystems, then this will significantly affect and impair the performance of fault detection.

Our approach as mentioned above does not require the modeling of system under test but instead just the system supplementary to system under test. We also

do not need to model the faulty behavior of system under test since we perform fault detection based on the expected behavior of system interaction and any discrepancies in the observed behavior will be classified as a fault. The problem of observability is also not in our consideration because fault detection is performed on system interaction which is entirely visible to us and so observability is not an issue. The success of out methodology relies on the system suitability and ability to be modeled as condition system. Condition system as a form of Petri net modeling formalism with explicit inputs and outputs namely condition signals allow us to represent system as a collection of subsystems which interact through the condition signals. One of the main advantages of using condition system Petri net is its ability to avoid state space problem in modeling of large and complex system such as state explosion. The well defined notions of input and output in condition system framework consequently allow the system model to exhibit clear cause & effect relationships. The dynamics within the subsystems can be defined independent of each other due to these well defined interfaces, and this would simplify the system model construction by allowing the reuse of subsystem models.

The fault detection of figure 3.1 is the more common fault detection terminology which is being used by most research communities. For example in quantitative, analytical redundancy methodologies and some DES researches [SFP03],[SSL96],[ZKW03]. In their approaches, given an actual system, they will study the system and generate an accurate system model from it. During the testing processes, inputs will be applied to both systems and the resulting outputs are compared. If the outputs discrepancy is off their fault-free limit/behavior, then failure is within the actual system. In their methodologies, the accuracy of system model representing the actual system is an issue.

Our fault detection scheme on the other hand does not require the modeling of actual system. In fact, we do not have the complete knowledge of the specification of the actual faulty system. All of our information about the actual system is obtained indirectly from the system supplementary to the actual system. Information is obtained through the signal interaction between the actual system (system under test) and supplementary system (auxiliary system) i.e. the input and output interaction between both systems. Therefore, we perform fault detection based on these I/O interactions, and this would mean that any legality that is beyond the interaction is out of our fault detection coverage. Therefore any legality which does

not cover in the system interaction is subjected to undetectability and the richness of system interaction among the system is an important issue. The richer the information convey from the system interaction, the more efficient our fault detection scheme will be.

In light of observation from the system behavior of I/O interaction within the actual system under test ($\mathcal{G}^{SUT,R}$) and auxiliary system of expected behavior ($\mathcal{G}^{Aux,E}$), there is a different type of fault in comparison to the type of fault which is detectable under our fault detection scheme. We are classifying faults into two different categories due to the fact that these two categories require slightly different fault detection methodology. Consider an outgoing signal from auxiliary system to system under test. This outgoing signal will represent a particular output that will influence the following command from system under test. Suppose that with this output signal from the auxiliary system, system under test should issue a command "A" to the auxiliary system. But due to some malfunction within system under test, the system issues a command "B" instead. Both command "A" and "B" are recognizable by the auxiliary system such that both incoming commands are within the legal language. Therefore, command "A" is a faulty behavior even though it is within the legality of the auxiliary system. Note that due to this new class of fault, expected language is different from legal language and the new fault is not within our current fault detection scheme coverage. A fault detection scheme for this type of fault is subjected to future research. In addition, we will also consider timed condition system formalism for our modeling and fault detection methodologies in our future research.

# Bibliography


[A04] Jeffery Ashley. Doctor of Philosophy Dissertation. DEPARTMENT OF ELECTRICAL ENGINEERING UNIVERSITY OF KENTUCKY, February 2004.

[CL99] Christos G. Cassandras and Stephane Lafortune. Introduction to Discrete Event Systems. KLUWER ACADEMIC PUBLISHERS, 1999.

[E03] Hartmut Ehrig. Petri net technology for communication-based systems: advances in Petri nets. SPRINGER-VERLAG NEW YORK LIMITED, 2003.

[GH00] Yu Gong and Lawrence E. Holloway. State Observer Synthesis for a Class of Condition Systems. IN IEE 5TH WORKSHOP DISCRETE EVENT SYSTEMS (WODES2000), Ghent, Belgium, August 2000.

[GH01] Yu Gong and Lawrence E. Holloway. Multi-layer State Observers for Condition Systems. DEPARTMENT OF ELECTRICAL ENGINEERING UNIVERSITY OF KENTUCKY, 2001.

[G85] T.H. Glisson. Introduction to System Analysis. MCGRAW-HILL, New York 1985.

[HA98a] Lawrence E. Holloway and Jeffery Ashley. Condition Languages and Condition Systems for Modeling Ambiguous Control Specifications. IN IEE INTERNATIONAL WORKSHOP DISCRETE EVENT SYSTEMS (WODES98), Cagliari, Italy, August 1998.

[HA98b] Lawrence E. Holloway and Jeffery Ashley. Elaborative Orderings of Condition Languages. IN PROCEEDINGS OF 1998 IEEE CONF. DECISION AND CONTROL, Tampa, Florida, December 1998.

[HA02] Lawrence E. Holloway and Jeffery Ashley. Diagnosis of Condition Systems Using Causal Structure. IN PROCEEDINGS OF THE AMERICAN CONTROL CONFERENCE, Anchorage, Alaska, May 2002.

[HCJK03] Z. Huang, V. Chandra, S. Jiang, and R. Kumar. Modeling Discrete Event Systems with Faults using a Rules Based Formalism. MATHEMATICAL AND COMPUTER MODELING OF DYNAMICAL SYSTEMS, 9(3), 2003.

[HGSA00] Lawrence E. Holloway, Xiaoyi Guan, Ranganathan Sundaravadivelu, Jeff Ashley, Jr. Automated Synthesis and Composition of Taskblocks for Control of Manufacturing Systems. IEEE TRANSACTIONS ON SYSTEMS AND CYBERNETICS-PART B: CYBERNETICS, 30(5), October 2000.

[KG95] Ratnesh Kumar, Vijay K. Garg. Modeling and control of logical discrete event systems. KLUWER ACADEMIC PUBLISHERS, 1995.

[MA98] John O. Moody, Panos J. Antsaklis. Supervisory control of discrete event systems using Petri nets. KLUWER ACADEMIC PUBLISHERS, 1998.

[O'C81] Patrick D.T. O'Connor. Practical Reliability Engineering. HEYDEN & SON LTD, 1981.

[P81] James L. Peterson. Petri Net Theory and the Modeling of Systems. PRENTICE HALL, INC., Englewood Cliff, New Jersey 1981.

[PP99] Charles L. Phillips and John M. Parr. Signals, Systems, and Transforms. PRENTICE HALL, INC., Upper Saddle River, New Jersey 1999.

[SFP03] Silvio Simani, Cesare Fantuzzi and Ron J. Patton. Model-based Fault Diagnosis in Dynamic Systems Using Identification Techniques. SPRINGER-VERLAG LONDON LIMITED, 2003.

[SK91] Ramavarapu S. Sreenivas and Bruce H. Krogh. On Condition/Event Systems with Discrete State Realizations. DISCRETE EVENT DYNAMIC THEORY AND APPLICATIONS, 1(2), September, 1991.

[SSL95] Meera Sampath, Raja Sengupta, Stephane Lafortune, Kasim Sinnamohideen and Demosthenis C. Teneketzis. Diagnosability of discrete event systems. IEEE TRANSACTION ON AUTOMATIC CONTROL, 40(9), 1995.

[SSL96] Meera Sampath, Raja Sengupta, Stephane Lafortune, Kasim Sinnamohideen and Demosthenis C. Teneketzis. Failure Diagnosis Using Discrete-Event Models. IEEE TRANSACTION ON CONTROL SYSTEMS TECHNOLOGY, 4(2), March, 1996.

[ZKW03] Shahin Hashtrudi Zad, Raymond H. Kwong and W.M. Wonham. Fault Diagnosis in Discrete-Event Systems: Framework and Model Reduction. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, 48(7), July 2003.

[ZD93] Meng Chu Zhou, Frank DiCesare. Petri net synthesis for discrete event control of manufacturing systems. KLUWER ACADEMIC PUBLISHERS, 1993.

[ZV99] Meng Chu Zhou, Kurapati Venkatesh. Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach. WORLD SCIENTIFIC, 1999.

# Vita

## Date and place of birth

January - 03 - 1980, Kuching, Malaysia.

## Educational institutions attended and degrees awarded

**B.S.E.E.**, University of Kentucky, Lexington, United States.

## Professional positions held

1. August 2003 – December 2004.

   **Research Assistant**, Department of Electrical Engineering, University of Kentucky.

## Scholastic and professional honors

1. August 2003.

   Awarded Research Assistantship to pursue M.S.E.E studies.

2. Januaury 2002.

   Awarded KGS scholarship to pursue M.S.E.E studies.

Andrew Hai Liang She