A Modeling and Analysis Methodology for Modular Logic Controllers of Machining Systems using Petri Net Formalism *

Euisu Park,[†] Dawn M. Tilbury,[‡] and Pramod P. Khargonekar[§]

NSF Engineering Research Center for Reconfigurable Machining Systems
2250 GG Brown, 2350 Hayward St.
University of Michigan
Ann Arbor, Michigan 48109-2125
email: {euisu, tilbury, pramod}@umich.edu

Submitted as a regular paper to the *IEEE Transactions on Systems, Man and Cybernetics*

Abstract

Logic controllers for machining systems typically have three control modes: auto, hand, and manual. In this paper, a unified formal representation of logic controllers with three control modes is provided using Petri nets. A modular logic controller structure is introduced and formalized for high-volume transfer lines. The modular logic controller consists of a control module for mode decision and control modules for station logic controllers. Each station control module is represented by connecting operation modules which are designed with respect to the fault recovery processes of operations; their connection algorithm is also provided. In our formal representation, each control module is represented by a live, safe, and reversible Petri net. A condition for the modular logic controller to generate a correct control logic is provided: operation causality condition. Using the modular structure of a logic controller, the control logic can be easily reconfigured and automatic code generation is possible.

*This work was supported by NSF ERC under Grant No. EEC9529125.

[†]Dept. of Electrical Engineering and Computer Science.

[‡]Dept. of Mechanical Engineering and Applied Mechanics.

[§]Dept. of Electrical Engineering and Computer Science.

1 Introduction

Automatic machining systems are designed for mass production of machined parts. These machining systems are controlled to automatically produce parts in repetitive cycles. In current industrial practice, this automation is achieved by sophisticated control systems involving logic control as well as servo control. The logic controller manages the sequencing of various processes to ensure that the desired operations are carried out. In addition to this automation control logic, the logic controller of a machining system is required to also handle exceptions such as fault stop/fault diagnosis and fault recovery because faults can happen in any operation for various reasons. In fact, industry folklore suggests that approximately 90% of the overall control logic is used for exception handling. Although the control logic for exception handling is important in a logic controller, there is no unified modeling method to represent the exception handling control logic with the automation control logic. As a result, it is not possible to conduct formal analysis and systematic design of the logic controller.

This work is partly motivated by the needs of the emerging class of machining systems called reconfigurable machining systems [1, 2]. In this context, machining systems and their logic controllers often need to be modified due to product part changes or process changes. It is important to be able to rapidly redesign and implement such logic controllers in response to mechanical reconfiguration of the machining system. Generally, the reconfigurability of the system results from the mechanical reconfigurability of the component machines. For a reconfigurable machining system, the logic controller for the system also should be reconfigurable. This is facilitated by having a modular representation for the component machines. A decentralized or a distributed logic controller has been considered for reconfigurable logic controllers and its hardware architecture and necessary functionality are introduced in [3, 4, 5].

Since logic controllers are event-driven supervisory systems, Petri nets are considered an appropriate tool for the analysis of the event-based behavior of logic controllers. A Petri net is a mathematical formalism and is also a graphical tool which consists of places, transitions, arrows, and tokens. By assigning physical meanings to each graphical component, Petri nets can model many systems which are characterized as being concurrent, asynchronous, distributed, and parallel. A major strength of Petri nets is their capability for analysis of important properties such as liveness, safeness, and reversibility associated with such systems. The general ideas of Petri net models and their application areas can be found in [6, 7, 8, 9]. For the details of Petri net models for logic controllers, see [10]. Frequently, the complexity of a real-world system requires a large Petri net having many places and transitions; the analysis of such a net system is limited by available computational resources. To overcome this complexity, simple reduction rules are developed and introduced in [6, 11]. Another way to overcome the complexity is to build a Petri net using

top-down or bottom-up approaches. The top-down approach is based on a step-by-step refinement and the bottom-up approach relies on the aggregation of subnets sharing common places or transitions; a simple and effective synthesis method is applied for modeling of manufacturing systems [12, 13, 14].

Many authors have investigated supervisory controllers in automated manufacturing systems using Petri net formalisms [15, 16, 17, 18]. One of the main issues in supervisory controllers is to build a Petri net model which realizes a given control logic while satisfying the required structural properties such as liveness, boundedness, and reversibility. In [19, 14], a CAD/CAE environment for this hierarchical design methodology was introduced. However, it seems that this approach is not yet much utilized in the machining industry. This is due in part to the facts that the methodologies are not simple (they require specialized formalisms about Petri nets) and the given control logic is too restricted to apply in real manufacturing systems. Most importantly, this method cannot be used directly in logic controllers for machining systems. In machining systems, logic controllers typically have three control modes: auto, hand, and manual. The control logic for hand and manual modes are designed for various fault recovery in machining systems. Even though fault recovery in automated manufacturing systems has been investigated in various ways [20, 21, 22, 16, 23], the control logic for the three control modes has not been considered at the same time.

The main contribution of this paper is to develop a logic control design methodology which includes the exception handling control logic along with the normal operation cycle in a modular structure. In our methodology, each operation in a logic controller is modeled first using Petri net formalism. Then, two types of operation modules are designed with respect to fault recovery characteristics of operations. To overcome control logic complexity due to different control modes, a superposition methodology is developed; the control logic for each control mode can be designed separately. Using internal variable conditions to account for the synchronization between stations, the control logic for each station in a machining system can be also designed separately. A modular logic controller is introduced; it consists of mode decision control module and station control modules. The correctness of control logic for a modular logic controller is verified using the properties of Petri nets and the operation causality condition. In most cases, logic controllers are implemented on PLC's (Programmable Logic Controllers). The designed modular logic controller can be implemented on PLC's directly by using SFC (Sequential Function Chart) via automatic code generation. Finally, we show that the modular logic controller can be easily reconfigured due to its modular structure.

The paper organized as follows. Section 2 describes the main characteristics of logic controllers for automatic machining systems. In Section 3, some basic definitions and notations for Petri nets are briefly introduced. The key ideas in modeling of logic controllers with three control modes are presented in Section 4. A modular logic controller and its condition to generate correct control logic

is developed in Section 5 and its application with an illustrative example is explained in Section 6. Finally, in Section 7, the results are summarized.

2 Logic Controllers for Automatic Machining Systems

Logic controllers for automatic machining systems control the operation sequences to achieve the goal of the machining system. Automatic machining systems are designed to produce a machined part automatically in every repetitive cycle; a number of machines are linked together to provide complete processing of a part. The automatic machining systems considered in this paper are high-volume transfer lines; the main characteristics of these machining systems and their logic controllers will be introduced in this section.

2.1 High-Volume Transfer Lines

Generally, high-volume transfer lines are composed of several machining stations together with mechanisms for part transferring and fixturing. A high-volume transfer line consists of a transfer mechanism, a number of machining stations, and fixturing mechanisms for the machining stations. A part is presented to each machining station by a transfer mechanism, and it is clamped by a fixturing mechanism. A number of asynchronous machining stations are synchronized by part flow. Each mechanism and machining station is simply called *a station*. Each station performs a particular process necessary to the overall production of the part using simple and fast motions. Each motion is called an *operation* and a sequence of operations is associated with each station. The schematic diagram of a high volume transfer line is shown in Figure 1. In this transfer line, all machining stations are milling stations and the motions in each machining station are provided by two or three single axis sliding mechanisms (vertical axes are not shown in the figure).

2.2 Controls in Machining Systems

In automatic machining systems, two distinct types of controllers coexist: sequence control and motion control. The logic controller of a machining system handles the sequencing operations such as part transfer, fixturing operations and coordination of machining stations. It also provides system status information and interacts with the operator for "manual" manipulation. In addition to this sequence control, some operations (such as the metal-removal operations) require motion control for precise feed rates and positioning. This motion control can be achieved by various feedback control algorithms for servomechanisms; these control algorithms are well developed and some of them can be found in [24, 25].

In high-volume transfer lines, motion control can be achieved by a network of computerized motion controllers (CMC's) [26]. These controllers are programmed using motion blocks in the EIA-

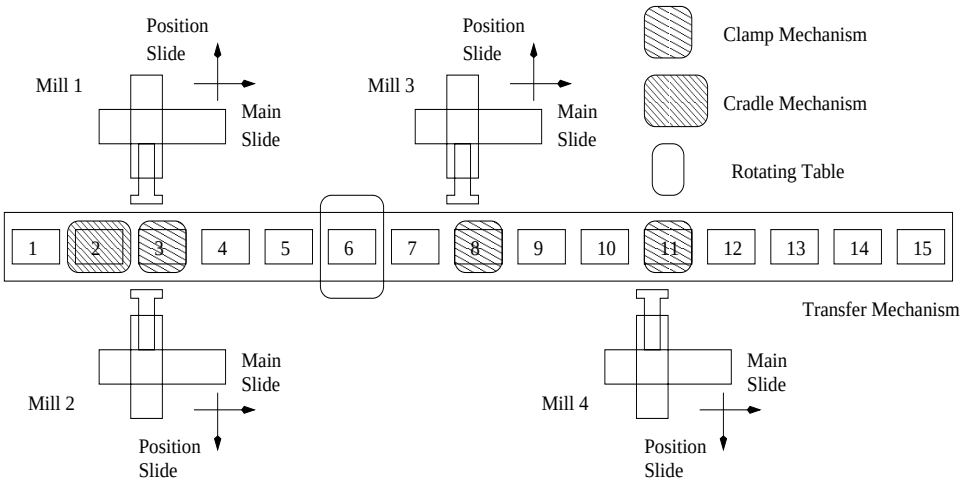


Figure 1: A high-volume transfer line for engine block surface milling

RS 274 format, which is similar to the G-code part program used in CNC (Computer Numerical Control) machines. A motion block or a sequence of motion blocks is considered as an operation in the logic controller. Some of these operations are causally related to operations in other stations. Typically, a motion block or a sequence of motion blocks is initiated by the logic controller, while the precise feed rates and positioning are achieved by the motion controller.

2.3 Control Modes in a Logic Controller

One of the main characteristics of a logic controller for an automatic machining system is its distinct control modes. Each control mode has its own control logic which is designed to achieve the functions for that particular control mode. The control logic for each mode is combined to form the system logic controller. Generally, a high volume transfer line has three control modes: auto, hand, and manual. The primary functions in each control mode will be described in this section.

2.3.1 Auto Mode

Operations are executed automatically in the auto mode. Only the logic controller is involved; operator commands are necessary only to initiate and stop this control mode. The main functions of auto mode are normal operation cycle, fault stop/fault diagnosis, and auto mode start/stop.

In a high-volume transfer line, the normal cyclic operation consists of the following sequential steps:

1. return clamp (Unclamp parts and clear fixturing mechanisms from the transfer bar path.)
2. raise transfer (Raise parts.)
3. advance transfer (Move parts to the next position.)

4. lower transfer (Lower parts.)
5. advance clamp (Clamp parts by fixturing mechanisms.)
6. cycle machining stations (Advance tool heads into parts and carry out machining operations at prescribed feed rates and spindle speed.)
7. return transfer (Return transfer bar to starting position.)

The time required to complete one cycle, called the cycle time, determines the number of parts that can be produced on a system, and thus is an important issue in high-volume transfer lines [27]. To reduce the cycle time, the sequential steps are overlapped in the sense that a new step may begin before the previous one has finished. This efficient cyclic operation is achieved by the control logic for the normal operation cycle. In current industrial practice, the normal operation cycle specification is given by a timing bar chart. A part of the timing bar chart for the system in Figure 1 is shown in Figure 2. The dotted arrows show causal dependencies between operations in different stations. The operations which are necessary for the normal operation cycle are called *normal operations*.

A fault can occur in any operation of an automatic machining system. Therefore each operation must have associated control logic for fault stop/fault diagnosis. The faults in operations can be detected by "watch-dog timers" in the logic controller or additional monitoring devices for tool breakage or belt breakage, etc. If a fault is detected during an operation, the logic controller stops the operation and announces the fault to the operator; this announcement is called fault diagnosis. A logic controller points out the fault exactly so an operator can easily find the cause. For this reason, a logic controller needs to trace the state of every operation. Because of the causal dependencies in the operations of a machining system, the entire machining system is stopped by any fault. If a fault happens in one station, the other stations are stopped just after finishing their current operations; new operations cannot be initiated. The system must wait for a re-start command from the operator to begin again. Another type of fault stop in an automatic machining system is E-stop (Emergency stop) which is initiated by the operator. In this case, the entire machining system is stopped immediately for safety.

The home position is the state of a machining system where the system stops when the system stop command is given by the operator. Each station has its own home position where its mechanisms are mechanically stable; the state where all stations are in their home positions is the home position of a machining system. For each station, the normal operation which moves the station to its home position is called the *home operation*; the home operations for each station are indicated in the timing bar chart by a $*$. The auto mode control logic can start the machining system either from the home position or after a fault recovery. Auto mode start is a transient control process before a machining system begins its normal operation cycle. The auto mode stop control logic

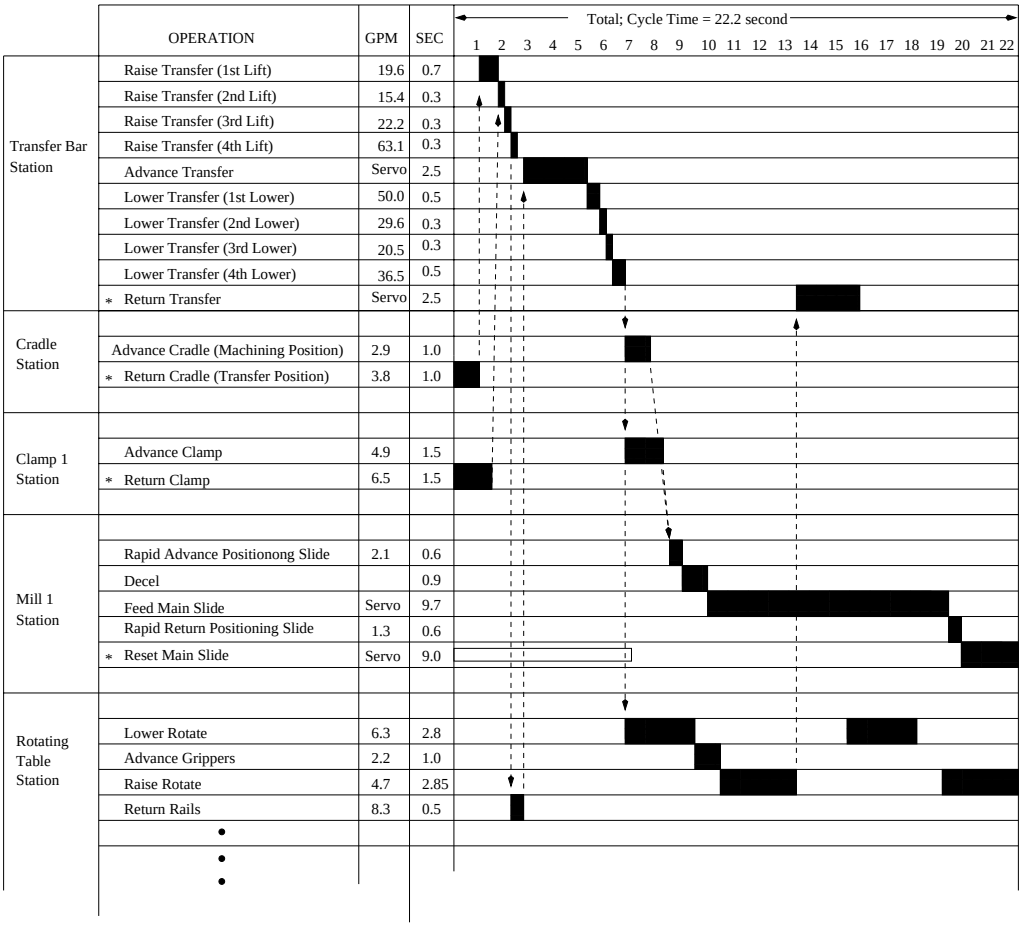


Figure 2: A section of the timing bar chart for the high-volume transfer line in Figure 1. The modules are listed in the left-most column. Each operation is listed, and the the time taken by each operation is indicated by a solid black bar. Operations which carry over to the next cycle time are indicated by a white bar. Causal relationships between operations are indicated by dotted lines and the home operation in each station is indicated by a $*$.

brings the machining system to the home position after the normal operation cycle is complete. Auto mode stop is another transient control process which must be included in the auto mode control logic.

2.3.2 Hand Mode

When an automatic machining system is running, the operator often needs to coordinate multiple stations or validate the automatic process step by step. Hand mode is used for these functions. In hand mode, the logic controller carries out the sequential steps introduced in Section 2.3.1 without the overlapping that is present in the normal operation cycle; one entire operation cycle is completed semi-automatically by a sequence of operator commands. An operation, a sequence of operations, or sequences of operations are assigned to each step; they are initiated when the corresponding operator command is received. Moreover, some steps are designed to be repeatable; operations which are initiated by one operator command can be reversed to the initial state by another operator command.

Generally, hand mode consists of seven to ten steps depending on the complexity in causal dependency of operations between stations. The input device for the hand mode operator commands is located in the main console of a machining system and will be discussed in more detail in Section 6 (see Figure 23).

2.3.3 Manual Mode

The main functions of the manual control mode are fault recovery and single station fine operation control (such as jog motions). If a fault (or faults) occurs in an operation (or operations), the entire machining system is stopped by the fault stop control logic. Faults in a machining system can only be recovered by the operator manually; automatic fault recovery is not possible in current industrial practice. After removing the cause for the fault, the operator re-starts the auto mode. The input devices for manual manipulation by the operator are located at each station. Generally, the manual control mode for fault recovery is executed by the operator commands *advance* and *return* (see Figure 18 (a)).

Sometimes, a station needs to finish its cycle manually or the station must go to a certain position to resume the automatic process properly. For this purpose, a sequence of operations in a station is designed to be run by the input device in each station. The manual advance operation follows the order of the sequence in the normal operation cycle for each station and the manual return operation follows the inverse order of the sequence.

2.4 Mode Switching of Automatic Machining Systems

Automatic machining systems are operated by the control logic for each control mode; each control mode can be initiated by the operator. A typical mode switching of automatic machining systems is shown in Figure 3. A machining system has stopped at its home position from the end of the previous operation. For machining systems, this state can be changed slightly due to leakages in stop valves of hydraulic mechanisms. Therefore, when the operator first starts the system, the machining system should be checked whether it is in the proper state. If not, the operator recovers the system using the jog motion of the manual control mode. If the system is ready, the operator can start the auto mode operation. If no faults occur, the manual and hand control modes are not needed. If a fault does occur, the machining system is stopped by the control logic for fault stop/fault diagnosis. The operator must recover the fault in the manual control mode. The machining system then goes back the auto mode either directly or after checking the other normal operations using hand mode. In hand mode operation, faults can also occur; the machining system is stopped again by the fault stop control logic. When the system stop command is received, the auto mode stop control logic stops the machining system at its home position. Thus, in addition to the individual mode control logic, the system logic controller must ensure that mode-switching is handled appropriately.

2.5 Configuration of a Logic Controller

Generally, logic controllers are implemented on PLC's (Programmable Logic Controllers), which are hardware devices with a CPU and memory. The CPU executes the control program; the memory is allocated for input, output, and internal variables. Input variables are assigned to sensors and operator commands; output variables are assigned to actuators and display devices for fault diagnosis. The logic controller we are considering in this paper has three different control modes; operations are executed in a different way in each control mode. To handle three control modes properly at the same time, a logic controller needs to keep track of the states of a machining system; internal variables are used for this. An internal variable is assigned to each normal operation; its value is changed by the control logic when the corresponding operation is completed. The details about the usage of internal variables will be explained in Section 4.3.

The operations of automatic machining systems are accomplished by actuators, mechanisms, and sensors. An actuator generates motion of a mechanism; both servo and hydraulic actuators are commonly used in machining systems. The position and the fault of a mechanism are detected by sensors. Limit switches, proximity switches, or encoders are used for position sensors; various fault sensors or watch-dog timers are used for the fault detection.

Operator commands are inputs to the logic controller for a machining system. The control

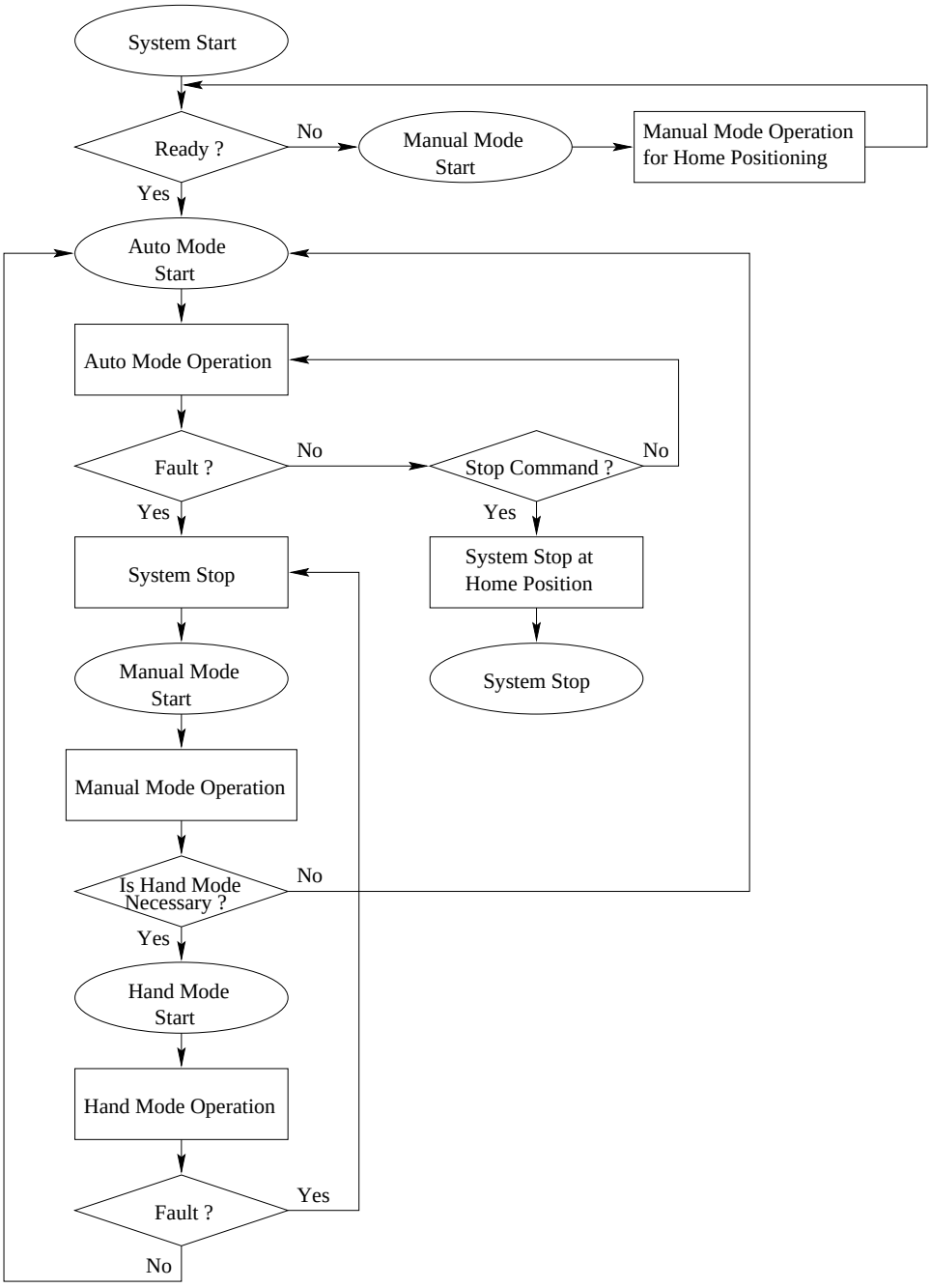
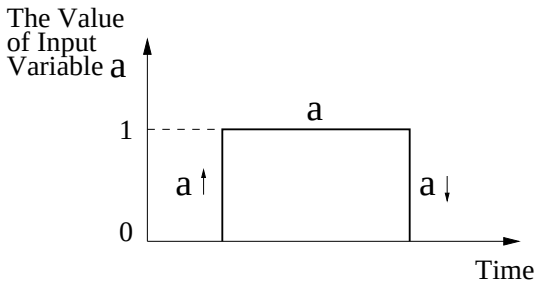


Figure 3: A typical mode switching of automatic machining systems



Representation	Events
a	active (a=1)
a↑	positive rising transition
a↓	negative falling transition

Figure 4: The events from each input variable: positive rising transition, negative falling transition, and active

mode of a logic controller is determined by operator commands. Operator commands also initiate operations in hand and manual control modes. Because operator commands come from input devices such as push buttons, it is possible an operator may request a command which is not allowed given the current state of a machining system. In this case, the logic controller must ignore the illegal command. The operator commands considered in this paper are control mode selection, the commands for each step in hand mode, and the advance/return commands in manual mode. Normally, the input devices for auto and hand modes are located at the main console of a machining system; the input device for manual mode is located at each station for fault recovery (see Figure 18 (a)). A boolean variable is assigned to each sensor or operator command input; it has 0 and 1 value. Therefore, three events can be generated from each input variable: positive rising transition, negative falling transition, and active (value equals 1). These three events are illustrated in Figure 4.

A configuration of a logic controller for an automatic machining system is shown in Figure 5. A logic controller consists of its control logic, input variables, output variables, and internal variables; it determines the values of output variables (normally, 0 or 1) with respect to the values of input and internal variables by its control logic.

2.6 Problem Statement

Mechanical engineers who design a machining system generate a timing bar chart which has the required cycle time and avoids mechanical conflicts. As mentioned above, a timing bar chart is the specification used in industry for the normal operation cycle of a machining system. However,

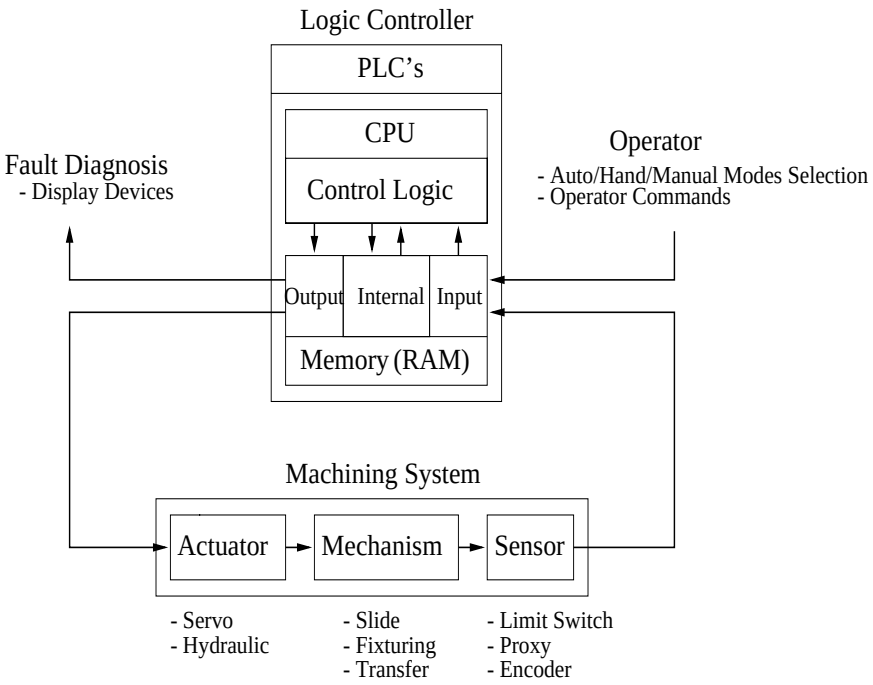


Figure 5: The configuration of a logic controller in an automatic machining system

the normal operation control logic is only a small fraction of the total logic controller. There are no formal specifications concerning the control logic for exception handling such as fault recovery. Thus, most of the control logic is programmed without any formal specifications; a logic controller strongly depends on the experience and knowledge of a programmer. This paper addresses the problem of developing a logic controller for a high-volume transfer line machining system using formal models of the control system, and incorporating all three control modes (auto, hand, and manual) as well as fault diagnosis and exception handling.

Even though the functions of logic controllers are very similar in high-volume transfer lines, programming the control logic approximately takes 50% of the total construction time for each machining system. Moreover, the control logic can be only verified in the "cycle and debug" stage, after the mechanism has been built. Although each machining system manufacturer has its own standard program generation techniques, the resulting logic controller program is very complicated and too large to be understood in a short time. One of main reasons for this complexity is that the control logic is programmed in an unstructured programming language known as ladder diagram. Ladder diagram has been the most popular programming language since the advent of PLCs due to its similarity to simple electrical relay circuit diagrams. Even though it is well known that ladder diagram is a very cumbersome way to represent sequence control, it is still widely used in most logic controllers for machining systems for historical reasons. Logic control programmers draw on their experience to program exception handling in ladder diagram; this experience does not exist

for other PLC languages. Although SFC (Sequential Function Chart) can be used for exception handling, its application is limited.

A formal modular representation of control logic using the Petri net formalism was proposed in [27]. However, only the control of normal operation cycle was addressed. The correctness of the control logic was verified using the properties of the Petri net model. In this paper, we show how the previously-proposed formal representation can be expanded to include control logic for all three control modes together with mode-switching logic. Again using the Petri net model, we show how key properties of the resulting control logic can be formally verified. We also introduce a modular design methodology for systematic control logic generation and implementation. In addition, we show how our design methodology can be a solution for logic controllers of reconfigurable machining systems.

3 Petri Nets

Logic controllers are event driven systems, and Petri nets are well known as a concise and rigorous formalism to model and analyze such systems. Moreover, the asynchronous and concurrent control behavior characteristics of logic controllers for machining systems can be easily represented by the Petri net formalism; the correctness of control logic can be also verified using Petri net properties. Some basic ideas of Petri net models for logic controllers are explained in [10, 28]. Petri nets have four graphical components: places (circles), transitions (bars), arcs (arrows), and tokens. The dynamic behavior of Petri nets can be achieved by tokens and the token firing rule. In this section, we present some background on Petri net formalism which will be used in this paper. More information on Petri nets can be found in [6, 7, 29, 11].

3.1 Petri Net Formalism

A marked Petri net is defined as a five-tuple $N = \{P, T, Pre, Post, M\}$ where P is the set of places, $P = \{p_1, p_2, \ldots, p_n\}$, $|P| = n$, T is the set of transitions, $T = \{t_1, t_2, \ldots, t_m\}$, $|T| = m$ with $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$, $Pre(Post)$ is the *pre-*(*post-*) *incidence* function representing the input (output) arcs, $Pre : P \times T \rightarrow \{0, 1\}$ ($Post : T \times P \rightarrow \{0, 1\}$), and M is the *n-dimensional* marking of the system, $M : P \rightarrow \{0, 1, 2, \ldots\}$; $M(p)$ is the *token count* of place p. There is an arc going from the place p_i to the transition t_j *iff* $Pre(p_i, t_j) = 1$. Similarly, there is an arc going from transition t_k to place p_i *iff* $Post(t_k, p_i) = 1$. The input and output sets of a transition $t \in T$ are represented by ${}^\bullet t = \{p \,|\, Pre(p, t) = 1\}$ and $t^\bullet = \{p \,|\, Post(t, p) = 1\}$ respectively. The input and output sets of a place $p \in P$ are represented by ${}^\bullet p = \{t \,|\, Pre(p, t) = 1\}$ and $p^\bullet = \{t \,|\, Post(t, p) = 1\}$ respectively. A marked Petri net with the given initial marking M_0 is denoted by $< N, M_0 >$.

A marking in a net system evolves according to the following firing rules: (1) A transition $t \in T$

is enabled iff for all p such that $Pre(p,t) = 1$, $M(p) > 0$ and (2) The firing of an enabled transition t is an instantaneous operation that removes a token from each input place p of t, and adds a token to each output place p of t. (3) Enabled transitions are never forced to fire. The firing rule (3) implies a form of nondeterminism. In practical modeling, this nondeterminism is resolved by the interpretation of a transition (*e.g.* a transition can be fired on the occurrence of an external event). The interpretation of transition firing for logic controllers will be addressed in Section 4.2. The notation $M_1[t\rangle M_2$ denotes that transition t is enabled at the marking M_1 and the marking M_2 is reached from M_1 by firing t. A sequence of transitions $\sigma = t_1, t_2, \ldots, t_k$ is called a firing sequence of $< N, M_0 >$ *iff* there exists a sequence of markings such that $M_0[t_1\rangle M_1[t_2\rangle M_2 \cdots [t_k\rangle M_k$. The set of all reachable markings from M_0 in a net $< N, M_0 >$ is denoted by $R(N, M_0)$.

For Petri net models for logic controllers, three important properties are typically considered: liveness, safeness, and reversibility.

Definition 1 *A marked Petri net $< N, M_0 >$ is live iff $\forall t \in T$, $\forall M \in R(N, M_0)$, there exists a sequence of transitions σ such that a marking reachable from M by firing σ, enables t. A marked Petri net $< N, M_0 >$ is safe iff $\forall p \in P$ and $M \in R(N, M_0)$, $M(p) \leq 1$. A marked Petri net $< N, M_0 >$ is reversible iff $\forall M \in R(N, M_0)$, $M_0 \in R(N, M)$.*

The physical meaning of these three properties for a logic controller can be summarized as follows [13]:

- Liveness as considered in this paper means the absence of deadlocks. This property guarantees that all transitions are firable and all the states of a logic controller which are modeled by places can occur.
- Safeness guarantees that there is no attempt to request execution of ongoing process. Another important implication of safeness is the Boolean representation of places; this enables a direct conversion from Petri net to Sequential Function Chart [30] which is a programming language for PLC's.
- Reversibility implies that the system will have a cyclic behavior and will perform its function repeatedly. It also characterizes the recoverability of the initial state from any state (in the controlled behavior) of the system.

We need following definitions to explain the structure of Petri nets.

Definition 2 *In Petri nets, a sequence of places and transitions, $p_1 t_1 p_2 t_2 \cdots t_{n-1} p_n$ ($t_1 p_1 t_2 p_2 \cdots p_{n-1} t_n$) is a directed path from p_1 to p_n (t_1 to t_n) if $t_i \in p_i^\bullet$ and $t_i \in {}^\bullet p_{i+1}$ for $1 \leq i \leq n-1$ (if $p_i \in t_i^\bullet$ and $p_i \in {}^\bullet t_{i+1}$ for $1 \leq i \leq n-1$). Moreover, if there are no repeated places or transitions in the sequence, a directed path is called simple.*

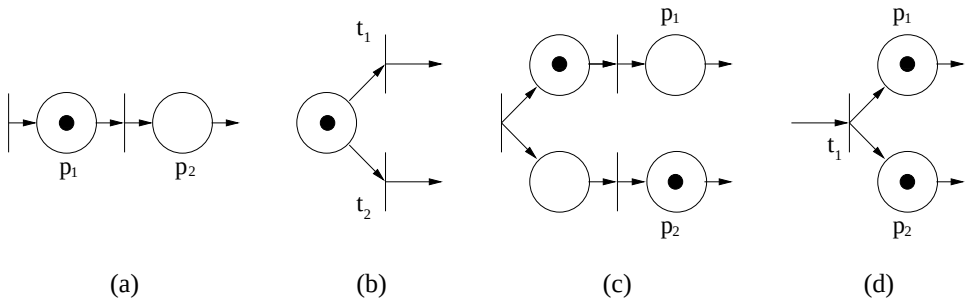


Figure 6: Basic modeling features of Petri nets: (a) causal dependence, (b) conflict, (c) concurrency, (d) synchronization

Definition 3 *In Petri nets, a directed path* $p_1t_1p_2t_2\cdots t_{n-1}p_n$ $(t_1p_1t_2p_2\cdots p_{n-1}t_n)$ *in which* $p_1 = p_n$ $(t_1 = t_n)$*, is a directed circuit. If the directed path is simple, the directed circuit is also called simple.*

Definition 4 *A state machine is a Petri net such that every transition has exactly 1 input and 1 output place,* $|\ {}^\bullet t\ | = |t^\bullet| = 1, \forall t \in T$.

Definition 5 *A Petri net is said to be strongly connected if there exists a directed path from every transition (or place) to every other transition (or place).*

Using graphical representations, Petri nets can easily model causal dependencies (sequences), conflict (decision, choice), concurrency, and synchronization, which are the main features of control logic for automatic machining systems. These modeling features are illustrated in Figure 6. Figure 6(a) shows the causal dependence between p_1 and p_2. The transitions in Figure 6(b), t_1 and t_2, are simultaneously enabled. If t_1 fires, t_2 is not enabled and vice versa (simultaneous firing is not allowed). This Petri net structure is called a conflict or choice. Figure 6(c) shows the concurrent behavior of p_1 and p_2. They are causally independent. Synchronizations in a distributed and concurrent systems can be modeled as in Figure 6(d). The places, p_1 and p_2, are synchronized by firing the transition t_1.

3.2 Petri Net Interpretation for Logic Controllers

A discrete event dynamic system can be modeled as a Petri net by assigning physical meanings to the graphical elements: places, transitions, and tokens. All physical meanings assigned to a transition will be simply called *conditions* of the transition. The mutually exclusive conditions must be assigned to the conflict transitions to ensure deterministic behavior [31, 32]. The state of a system can be represented by the marking M of a Petri net; in a logic controller, the sequence of control actions is generated by the dynamic evolution of Petri net model.

In the Petri net model for a logic controller, the firing rule for transitions needs to be modified to resolve the nondeterminism of Petri nets.

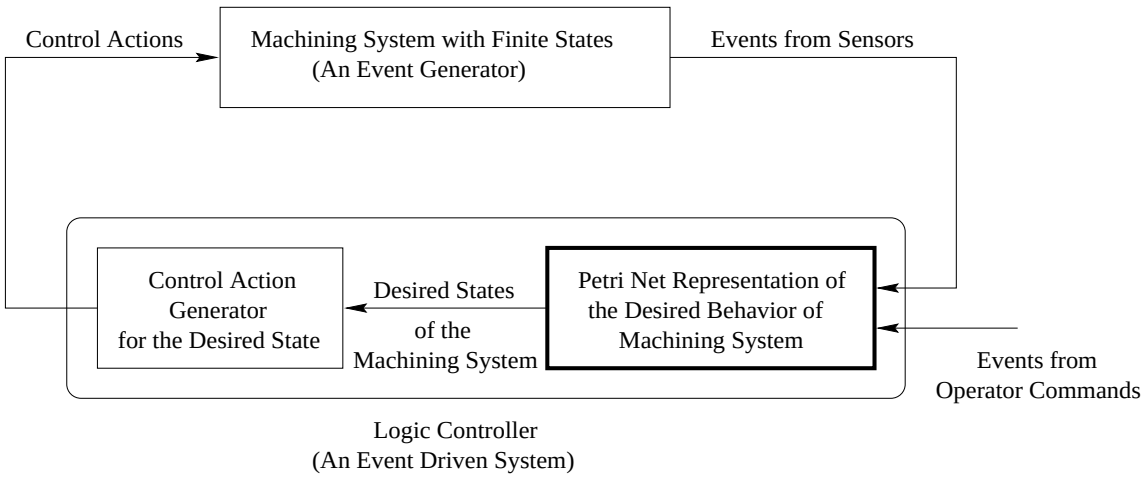


Figure 7: The schematic diagram of the closed loop system for a machining system with its logic controller

Definition 6 (Firing rule of the Petri net model for a logic controller) *Enabled transitions are fired instantaneously as soon as the conditions assigned for transitions are satisfied.*

4 Modeling of Logic Controllers using Petri Net Formalism

A logic controller controls the on/off actions of all the operations in its machining system; the control logic should be designed to generate proper control actions for operations with respect to given inputs from sensors or operator commands. In this section, key ideas in modeling of logic controllers for automatic machining systems will be introduced.

4.1 A Supervisory Controller

The schematic diagram of the closed loop system with a machining system and its logic controller is shown in Figure 7. The logic controller is an event-driven system. It can generate the control actions from the desired states information provided by the Petri net model. The desired behavior of a machining system is represented by the Petri net model; a logic controller which has the desired behavior of the controlled system is called a supervisory controller [33]. By assigning proper control actions to the states (i.e. places) of the Petri net model, a logic controller can generate control actions whenever the places are activated by tokens (as in Moore-Automata [34]). The machining system is driven by the control actions and generates events; it can be considered an event generator.

This paper introduces a Petri net based model of the logic controller represented by the bold lined box in Figure 7. The logic controller Petri net models the *desired closed-loop behavior* of the machining system; the properties of the closed-loop system will be determined by analyzing

this model. As mentioned in Section 3.1, the properties required in closed loop systems with logic controllers are liveness, safeness, and reversibility. The control actions of a logic controller are generated by the state evolution of the Petri net model. Therefore, the correctness of the control logic can be verified by analyzing the properties of the Petri net model.

4.2 Petri Net Block for an Operation

A machining system must perform a variety of operations. Within the logic controller, each operation must have a Petri net representation. In machining systems, an operation has the following characteristics:

- an operation can only be initiated by the logic controller.
- faults can occur during the execution of an operation.
- an operation can be stopped in either a completed or incomplete state.

An internal variable is assigned to each operation to keep track of its proper completion. The control actions necessary for an operation are activate/deactivate the corresponding actuator, fault diagnosis, and internal variable manipulation.

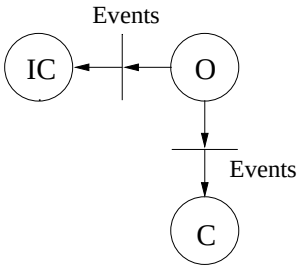
Any controlled operation in a machining system can be represented by three states: *operating*, *completed stop*, and *incomplete stop*. In Petri nets, each state can be modeled by a place. Therefore, each operation can be represented by three places in the Petri net model for a logic controller, i.e.,

$$P(\mathcal{O}) = \{p_O^{\mathcal{O}},\ p_{IC}^{\mathcal{O}},\ p_C^{\mathcal{O}}\} \tag{1}$$

where $P(\mathcal{O})$ is the set of places for the operation $\mathcal{O}$ and the subscripts O, IC, and C represent the operating, incomplete stop, and completed stop states respectively. By assigning the appropriate control actions to each state (place), and linking the states using transitions and arrows, an operation can be formally represented by a Petri net. Control actions for the operating state, completed state, and incomplete state are *activate the corresponding actuator*, *deactivate the corresponding actuator and set the corresponding internal variable 0 to 1*, and *deactivate the corresponding actuator and fault announcement of the operation* respectively. A Petri net block for an operation with the corresponding control actions is shown in Figure 8.

4.3 Internal Variable Conditions

In machining systems, the operations in each station are executed concurrently and for the most part asynchronously. However, to prevent mechanical conflicts or reduce cycle time, there may be causal relationships between operations in different stations. These causal relationships in the normal operation cycle are represented by dotted arrows in the timing bar chart (see Figure 2).



Operation States	Control Actions
O (operating)	activate the corresponding actuator
C (completed)	deactivate the corresponding actuator and set the corresponding internal variable 0 to 1
IC (incomplete)	deactivate the corresponding actuator and fault announcement of the operation

Figure 8: A Petri net block for an operation with the corresponding control actions : O, C, and IC represent operating, completed, and incomplete state of an operation respectively.

In [27], we represented these causal relationships using synchronized transitions. However, these causal relationships depend on the control mode, and cannot be used if all three control modes are considered at the same time. Instead of synchronized transitions, we use internal variables to represent the completion of operations, and share this internal variable information between stations. By including internal variables in transition conditions, the causal relationships of operations in different concurrent sequences can be achieved. Consequently, the control logic for each station in a machining system can be represented separately.

As mentioned in Section 2.5, the control logic generates the control outputs (on/off actions) using the event information of input variables and internal variables. The transition conditions in Petri nets represent this information. In our model, the internal variable conditions are represented by a separate arrow to distinguish them from input variable conditions. Figure 9 shows how these conditions are represented in a transition.

Some causal relations which are commonly found in machining systems are illustrated in Figure 10 with their internal variable condition representations. Figure 10 (a) shows a causal dependence between operations $A1$ and $B2$; operation $B2$ can start after completing operations $B1$ and $A1$. The representation with the internal variable condition $IVC\ A1$ is also shown; $IVC\ A1$ implies that the internal variable assigned to the operation $A1$ is set to 1 (completed). These two representations are equivalent. Figure 10 (b) represents a synchronization of the operations $A2$ and $B2$. Using internal variable conditions, two concurrent and synchronized sequences can be represented separately.

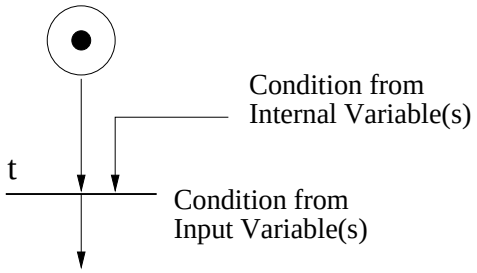
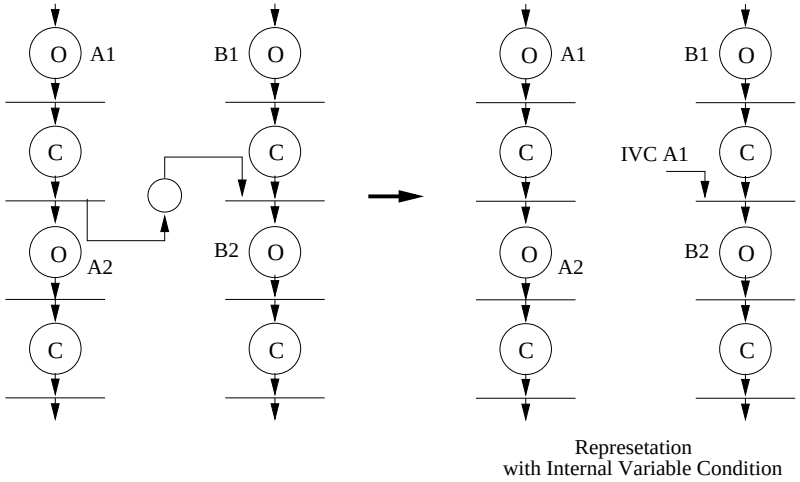
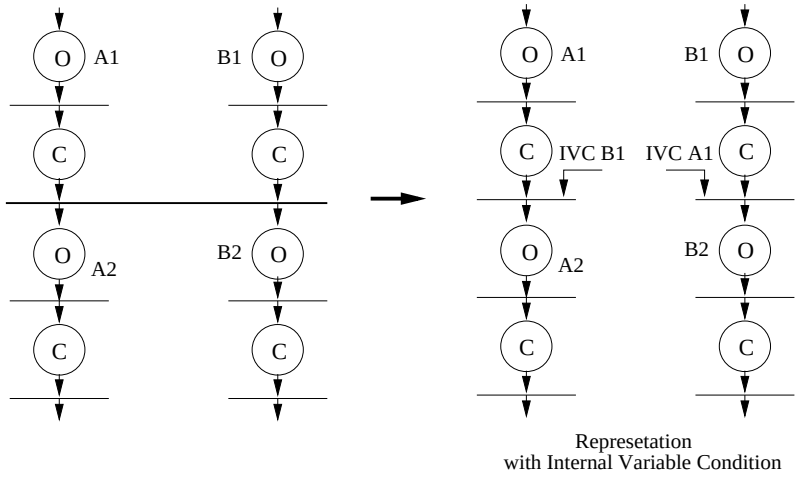


Figure 9: The graphical representation of conditions for a transition: internal variable conditions and input variable conditions



(a) causal dependence



(b) synchronization

Figure 10: The representations of causal relations of operations in concurrent operation sequences

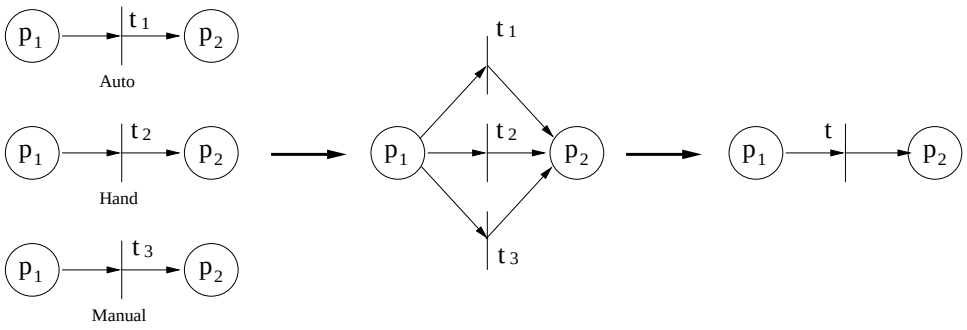


Figure 11: The superposition design methodology: $\mathcal{C}_t = (auto \wedge \mathcal{C}_{t_1}) \vee (hand \wedge \mathcal{C}_{t_2}) \vee (manual \wedge \mathcal{C}_{t_3})$

4.4 Superposition

A logic controller must handle all three control modes for an automatic machining system. Although each mode offers different functionality, they control the same set of operations. Thus the Petri net for each control mode is constructed by connecting the same set of places using transitions and arrows in such a manner that the specifications for that mode are met. The transitions may be different in each mode, or a common transition may have different conditions in different modes. The complete control logic can be obtained by combining the Petri nets for different control modes using superposition. Because the places are the same, they are superposed directly. The transitions for each control mode are superposed in the following way. Let $m1$ and $m2$ represent the different control modes mode 1 and 2. The transitions for the corresponding control mode are t_i^{m1} and t_j^{m2} respectively. If ${}^\bullet t_i^{m1} = {}^\bullet t_j^{m2}$ and $t_i^{m1\bullet} = t_j^{m2\bullet}$, the transitions t_i^{m1} and t_j^{m2} are common transitions. These common transitions are combined into one transition, t, with the conditions $\mathcal{C}_t$:

$$\mathcal{C}_t = (mode1 \wedge \mathcal{C}_{t_i^{m1}}) \vee (mode2 \wedge \mathcal{C}_{t_j^{m2}}) \tag{2}$$

where $\mathcal{C}_{t_i^{m1}}$ and $\mathcal{C}_{t_j^{m2}}$ are the transition conditions for mode 1 and mode 2 respectively and $\wedge$ and $\vee$ mean logical *and* and *or* respectively. This transition superposition is a well-known reduction rule: *fusion of parallel transitions*; Petri net properties such as liveness, safeness, and reversibility are preserved after this superposition. A proof of this property can be found in [35].

The superposition method is illustrated in Figure 11; $\mathcal{C}_{t_1}$, $\mathcal{C}_{t_2}$, and $\mathcal{C}_{t_3}$ represent conditions in auto, hand, and manual control mode for a common transition respectively. By superposition, a logic controller can be simply represented by Petri nets using the same set of places which can realize the control logic of three control modes.

4.5 Operation Modules

An advantage of the modular structure for a system is that the system can easily be built by connecting its modules. Moreover, the system can be easily modified partially without affecting the overall structure. In reconfigurable machining systems, machining stations can be reconfigured by changing their component mechanisms [36, 37]. In this case, some operations are added or replaced in the existing sequence of operations of a station. Therefore, a modular structure for control logic is necessary to handle the operation modifications; two modular representations for operations will be introduced here.

4.5.1 Types of Operation Modules

Because faults can happen at any time in a machining system, a logic controller should have fault recovery control logic for every operation. A fault stop in an operation can be recovered by one of three methods.

1. re-start the operation from the fault stop position directly.
2. return the mechanism to the initial position of the operation then re-start the operation.
3. return the mechanism to the home position of the station then resume the normal operation cycle of the station.

Operations in automatic machining systems can be categorized by their fault recovery methods. A normal operation whose faults can be recovered by method 1 or method 2 is defined as a *reversible operation*; a normal operation whose faults can be recovered by method is 1 or method 3 defined as an *irreversible operation*. Typically, motion-only operations such as "advance transfer" and "advance clamp" operations are reversible operations. However, the metal-removal operations can be reversible or irreversible depending on the nature of the metal-removal process. For example, drilling operations are reversible operations; surface milling feed operations are irreversible operations (moving the mill backwards over the machined surface could damage the part). Therefore, two types of operation modules can be considered for the operations in machining systems: reversible operation modules and irreversible operation modules. The operations used for fault recovery of normal operations are called *fault recovery operations*. Because the normal operations in each station constitute a cyclic behavior, fault recovery operations are typically the same as normal operations. However, we consider the fault recovery operations to be different operations because they are frequently executed with different feed rates than their corresponding normal operations. Each reversible operation has a corresponding fault recovery operation. Generally, faults in irreversible operations are recovered by a sequence of other reversible operations.

4.5.2 Petri Net Representation

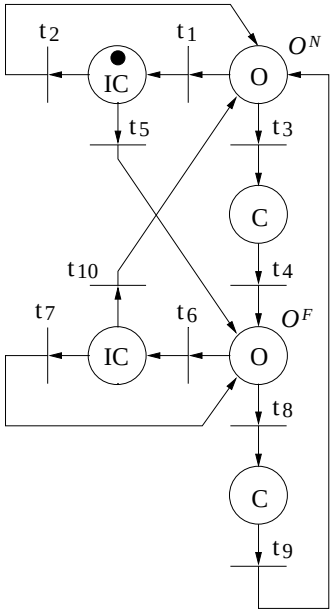
Here we show how the reversible operation module (R-module) and the irreversible operation module (I-module) can be represented using Petri nets. Because faults can only be recovered in manual mode, the Petri net representations are developed using manual mode events, namely the advance and return operator command inputs. An R-module consists of two Petri net blocks: one for the normal operation and the other for its fault recovery operation. An I-module is represented by a Petri net block for a normal operation; no corresponding single fault recovery operation exists. Let $\mathcal{M}_R^{\mathcal{O}}$ and $\mathcal{M}_I^{\mathcal{O}}$ represent an R-module and an I-module respectively. Then, the Petri nets for two operation modules consist of the following sets of places.

$$P(\mathcal{M}_R^{\mathcal{O}}) = P(\mathcal{O}^N) \cup P(\mathcal{O}^F) = \{p_O^{\mathcal{O}^N},\ p_{IC}^{\mathcal{O}^N},\ p_C^{\mathcal{O}^N},\ p_O^{\mathcal{O}^F},\ p_{IC}^{\mathcal{O}^F},\ p_C^{\mathcal{O}^F}\} \tag{3}$$

$$P(\mathcal{M}_I^{\mathcal{O}}) = P(\mathcal{O}^N) = \{p_O^{\mathcal{O}^N},\ p_{IC}^{\mathcal{O}^N},\ p_C^{\mathcal{O}^N}\} \tag{4}$$

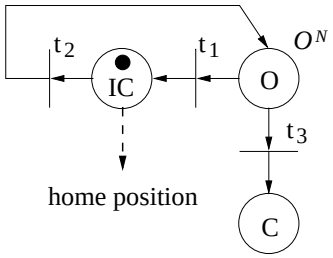
where $P(\mathcal{M}_R^{\mathcal{O}})$ and $P(\mathcal{M}_I^{\mathcal{O}})$ are the sets of places for an R-module and an I-module. The superscript N and F represent the normal operation block and the fault recovery operation block of the operation $\mathcal{O}$ respectively. The places in each operation module are connected by fault recovery control logic for the operation. Figure 12 shows the representation of an R-module with the transition conditions for the operator commands in manual control mode; the initial marking represents the fault stop state of the operation $\mathcal{O}$. The variables a and r represent the input variables which are assigned to the operator commands in manual control mode: advance and return. As mentioned in Section 2.5, each operator command generates three events for its logic controller: positive rising transition, negative falling transition, and active (value equals 1). The Petri net structure is symmetric because faults can happen even in the fault recovery operation. In a fault recovery operation, the control action for completed state should be changed into *deactivate the corresponding actuator and re-set the corresponding internal variable 1 to 0* to indicate that its normal operation is initialized. The transitions t_3 and t_8 need the events from sensors in addition to operator command input for completion of each operation; s_N and s_F represent the event signals. The structure of an R-module can represent the control logic for fault recovery, jog motion, and operation repeat in manual control mode. The simple directed path $p_{IC}^{\mathcal{O}^N}\ t_2\ p_O^{\mathcal{O}^N}$ represents a fault recovery process: re-start the operation from the fault stop position directly. The simple directed path $p_{IC}^{\mathcal{O}^N}\ t_5\ p_O^{\mathcal{O}^F}\ t_8\ p_C^{\mathcal{O}^F}\ t_9\ p_O^{\mathcal{O}^N}$ represents another fault recovery process: return the mechanism to the initial position of the operation then re-start the operation. Moreover, jog motion and operation repeat can be also achieved in this Petri net structure.

The I-module is shown in Figure 13 with its transition conditions for the operator commands in manual mode; the initial marking represents the fault stop state of an irreversible operation $\mathcal{O}^N$. As in the R-module, the transition t_3 needs the event s_N from the sensor to indicate a completion



Transitions	Conditions	Transitions	Conditions
t_1	a$\downarrow$	t_6	r$\downarrow$
t_2	a$\uparrow$	t_7	r$\uparrow$
t_3	$s_N \wedge (\mathrm{r} \vee \mathrm{a})$	t_8	$s_F \wedge (\mathrm{r} \vee \mathrm{a})$
t_4	r$\uparrow$	t_9	a$\uparrow$
t_5	r$\uparrow$	t_{10}	a$\uparrow$

Figure 12: The Petri net representation of the R-module with the transition events from the operator commands in manual control mode. The variables a and r represent the advance and return operator commands in manual mode; their rising or falling transitions are indicated by arrows. S_N and S_F represent the sensor signals for the completion of the normal and fault recovery operations respectively. Logical *and* and logical *or* are indicated by $\wedge$ and $\vee$ respectively.



Transitions	Conditions
t_1	a$\downarrow$
t_2	a$\uparrow$
t_3	$s_N \wedge$a

Figure 13: The Petri net representation of the I-module with its transition events from the operator commands in manual mode. The variable a represents the manual advance command (there is no manual return for an irreversible operation); rising and falling transitions are indicated by arrows. S_N represents the event signal for the completion of the normal operation; $\wedge$ represents logical *and*.

of the operation in addition to the operator command input. Because faults of the operation $\mathcal{O}^N$ can be only recovered by executing a sequence of operations which returns the machining station to its home position, this operation module does not have its own fault recovery operation. One directional jog motion is possible in this module by the simple directed path, $t_2\ p_O^{\mathcal{O}^N}\ t_1\ p_{IC}^{\mathcal{O}^N}$.

5 Formal Representation of Modular Logic Controller

The control logic for each station in an automatic machining system station is considered to be a control module for the logic controller. The control logic which realizes the functions of all three control modes for a station is defined as a *station logic controller*. The control mode of a machining system is determined from the operator commands. Therefore, a logic controller with three control modes needs an additional control module for mode decision logic. In this section, a modular Petri net representation for the logic controller with three control modes will be developed.

5.1 Modular Logic Controller

The logic controller for a machining system consists of the mode decision control logic together with the station logic controllers. Causal relations with operations in other stations are handled by internal variable conditions. No explicit connections exist between these Petri nets; the control logic is modular. Each Petri net is called a control module of a logic controller; the control module

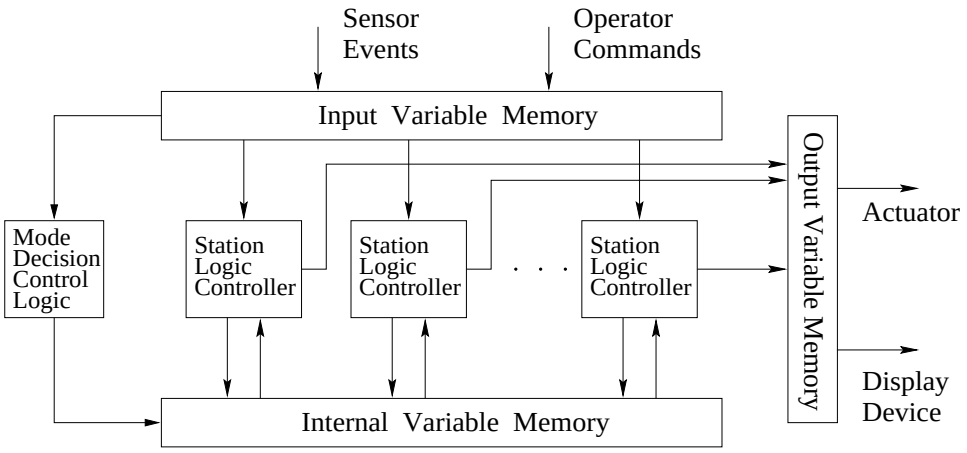


Figure 14: The configuration of the modular logic controller

for a station logic controller is represented by the operation modules and their connections. This modular representation can be defined as follows.

Definition 7 (Modular Logic Controller) *A modular logic controller is a formal Petri net representation of the logic controller for an automatic machining system which consists of one control module for the mode decision control logic and a station logic control module for each station. A system with n stations has $n+1$ control modules. In the control module for a station logic controller, the initial place marked with a token represents the home position of the station. Likewise, in the mode decision control module, the initial place marked with a token represents the home position of the machining system.*

The configuration of the modular logic controller is shown in Figure 14; the correlations between control modules are achieved through the internal variable shared memory. In this section, a formal method to construct the Petri net model for each control module along with analysis of its structural properties will be introduced. Moreover, a condition for the modular logic controller to generate correct control logic will be presented.

5.2 Station Logic Controllers

Each station logic controller must control the sequential behavior of the operations in that station in all three control modes, and keep track of its internal variables. It can be represented by a separate Petri net with internal variable conditions encoding the dependencies between stations. The operation sequences in a station logic controller are represented by appropriately connecting the operation modules described in the previous section.

5.2.1 Connection Algorithm for Operation Sequences

In each station, three types of operation sequences are possible: normal sequence, reverse sequence, and fault recovery sequence. Some normal sequences of reversible operations need to be reversed to their initial state for the repeatable steps in hand mode; the reverse sequence is necessary for this. If each operation is represented by its operation module, only the fault recovery sequences for irreversible operations (which are represented by I-modules) need to be considered because the fault recovery sequences for reversible operations are considered in the operation modules (R-modules). Some reversible operations in a station are initiated concurrently; they can be modeled by an R-module with a hierarchical structure. The hierarchical Petri net representation for concurrent operations is given in the Appendix.

A station can be considered an ordered sequence of operations; the ordering is derived from the timing bar chart and starts with the home operation, which may not be the first operation listed.

$$\mathcal{S}_j = \mathcal{O}_{1,j}, \mathcal{O}_{2,j}, \ldots, \mathcal{O}_{n_j,j} \tag{5}$$

where $\mathcal{S}_j$ and $\mathcal{O}_{i,j}$ are the jth station and the ith operation of jth station, respectively. The number of normal operations in the jth station is n_j and $\mathcal{O}_{1,j}$ is the home operation. Each operation in a station can be represented by its operation module, $\mathcal{M}_R^{\mathcal{O}}$ or $\mathcal{M}_I^{\mathcal{O}}$, depending if the operation is reversible or irreversible. The set of places for each module is described by Equations 3 and 4. Next we present a systematic procedure to construct the complete station logic control structure.

Step 1 *Assign an operation module for each operation listed in the timing bar chart.*
Use R-modules for reversible operations and I-modules for irreversible operations. The set of places in the Petri net for the station $\mathcal{S}_j$ is the union of the places in each module

$$P(\mathcal{M}^{\mathcal{S}_j}) = \bigcup_{i=1}^{n_j} P(\mathcal{M}^{\mathcal{O}_{i,j}})$$

where $P(\mathcal{M}^{\mathcal{O}_{i,j}})$ is the set of places in the ith operation module, which is either $\mathcal{M}_R^{\mathcal{O}}$ or $\mathcal{M}_I^{\mathcal{O}}$.

Step 2 *Create the normal operation cycle.*

Connect $p_C^{\mathcal{O}_{i,j}^N}$ and $p_O^{\mathcal{O}_{i+1,j}^N}$ with a transition for $i = 1$ to n_j, $n_j + 1 = 1$. This connection creates a simple directed circuit whose places consist of the operating and completed states of the normal operation blocks ($p_O^{\mathcal{O}_{i,j}^N}$ and $p_C^{\mathcal{O}_{i,j}^N}$) for all operation modules. The control logic for the normal operation cycle is represented by this simple directed circuit.

Step 3 *Create the reverse sequences for the repeatable steps in hand mode.*

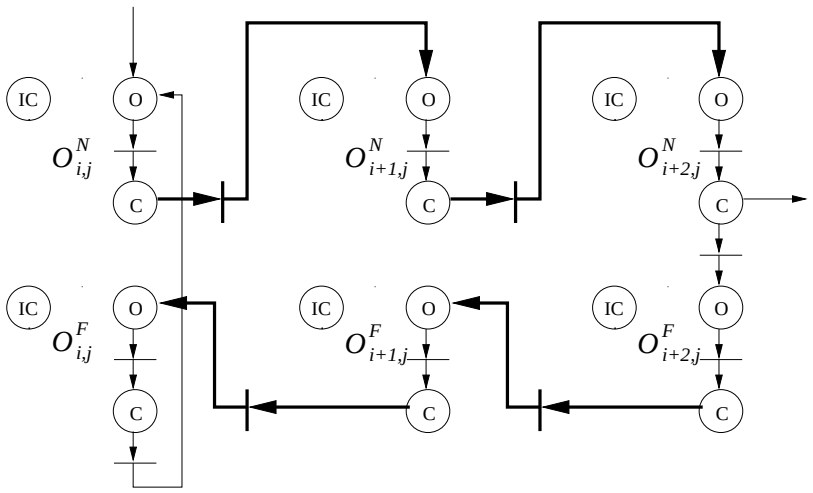


Figure 15: The connection for the reverse sequences: a normal sequence $\mathcal{O}^N_{i,j}, \mathcal{O}^N_{i+1,j}, \mathcal{O}^N_{i+2,j}$ and its reverse sequence $\mathcal{O}^F_{i+2,j}, \mathcal{O}^F_{i+1,j}, \mathcal{O}^F_{i,j}$ are represented by the bold arrows and the directed circuit with places $p_O^{\mathcal{O}^N_{i,j}}$,$p_C^{\mathcal{O}^N_{i,j}}$,$p_O^{\mathcal{O}^N_{i+1,j}}$,$p_C^{\mathcal{O}^N_{i+1,j}}$,$p_O^{\mathcal{O}^N_{i+2,j}}$,$p_C^{\mathcal{O}^N_{i+2,j}}$,$p_O^{\mathcal{O}^F_{i+2,j}}$,$p_C^{\mathcal{O}^F_{i+2,j}}$,$p_O^{\mathcal{O}^F_{i+1,j}}$,$p_C^{\mathcal{O}^F_{i+1,j}}$, $p_O^{\mathcal{O}^F_{i,j}}$,$p_C^{\mathcal{O}^F_{i,j}}$.

When a single operator command in hand mode initiates a sequence of reversible operations, and there exists another single operator command which brings this sequence of reversible operations to its initial position, a reverse sequence must be implemented. For every normal sequence, $\mathcal{O}_{i,j}, \mathcal{O}_{i+1,j}, \ldots, \mathcal{O}_{i+\alpha,j}$, $1 \leq \alpha \leq n_j - i$, for which a reverse sequence is necessary, connect $p_C^{\mathcal{O}^F_{i+\alpha-k,j}}$ and $p_O^{\mathcal{O}^F_{i+\alpha-k-1,j}}$ with a transition for $k = 0$ to $\alpha - 1$. By connecting the completed and operating states in the fault recovery operation blocks ($p_C^{\mathcal{O}^F_{i,j}}$ and $p_O^{\mathcal{O}^F_{i,j}}$) of the corresponding operation modules, the control logic for a reverse sequence can be represented. With its normal sequence, the control logic for a repeatable step in hand mode can be achieved by a simple directed circuit. This connection is illustrated in Figure 15.

Step 4 *Create the fault recovery sequence for irreversible operations.*
For each I-module $\mathcal{M}^{\mathcal{O}_{i,j}}$, construct a directed path from the incomplete stop state of the irreversible operation $p_{IC}^{\mathcal{O}^N_{i,j}}$ to the home operation of the station S_j ($p_O^{\mathcal{O}^N_{1,j}}$) via the operating and complete states (p_O and p_C) of the fault recovery operation block(s) in some of the previous operation modules $\mathcal{M}^{\mathcal{O}_{\beta,j}}$, for some β, $1 < \beta < i$. In general, not all operations must be reversed to get back to the home state. The sequence of the simple directed path is as follows:

$$p_{IC}^{\mathcal{O}^N_{i,j}}, \underbrace{\ldots p_O^{\mathcal{O}^F_{\beta,j}}, p_C^{\mathcal{O}^F_{\beta,j}}, \ldots}_{\text{for some } \beta, 1<\beta<i}, p_O^{\mathcal{O}^N_{1,j}}, p_C^{\mathcal{O}^N_{1,j}}$$

A typical operation sequence of a machining station is positioning, machining for metal-removal,

and re-positioning to the home position. The positioning process is accomplished by a sequence of reversible operations; the recovery sequences of I-modules can be achieved using some fault recovery operations of the positioning operations. Generally, all I-modules in a machining station are recovered by the same operation sequence.

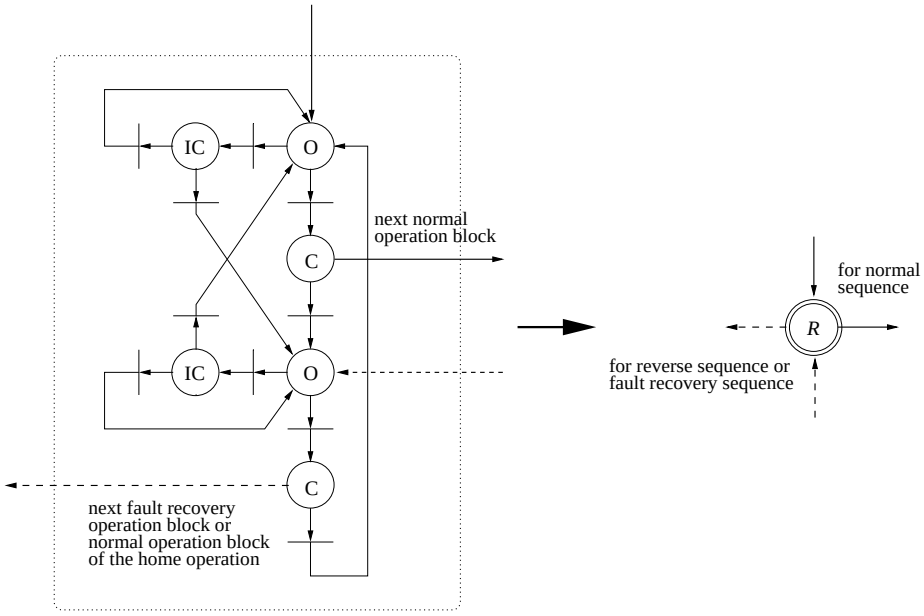
Each station logic controller is constructed by connecting its operation modules according to the above steps. If a station has j reversible operations and k irreversible operations, its logic control module has $6j + 3k$ places. All the necessary sequences in a station are achievable in the manual mode. Moreover, every transition in the operation modules has a manual mode condition. Therefore, all the transitions in the Petri net for a station logic controller should have conditions for manual control mode.

5.2.2 Hierarchical Representations for Station Logic Controllers

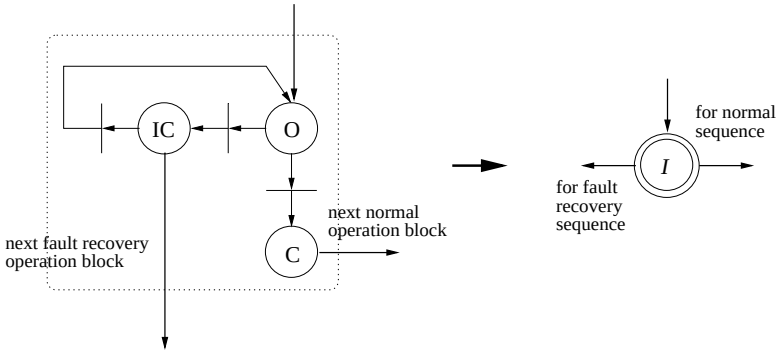
In our formal representation, the Petri net structure for a station logic controller is determined by the operation modules in a station and their connections as outlined in Section 5.2.1. Because the structure of an operation module is not changed when it is connected to other operation modules, each operation module can be represented as a single place with a hierarchical structure. This hierarchical place is denoted by a double circle; the hierarchical structures for an R-module and an I-module are illustrated in Figure 16. From the connection algorithm in the previous section, each R-module is connected to other operation modules from the completed places of its operation blocks ($p_C^{\mathcal{O}_{i,j}^N}$ and $p_C^{\mathcal{O}_{i,j}^F}$) as described by the operation sequence specifications, and each I-module is connected from its completed place ($p_C^{\mathcal{O}_{i,j}^N}$) and incomplete place ($p_{IC}^{\mathcal{O}_{i,j}^N}$). A dashed arrow is used in the R-module because the reverse sequence and fault recovery sequence in R-modules do not always occur.

Typically, two types of station logic controllers occur in automatic machining systems: station logic controllers with R-modules only and station logic controllers with R-modules and I-modules. Those types of station logic controllers are defined as R-stations and I-stations, respectively. The Petri nets for station logic controllers can be represented simply by using the hierarchical structures of operation modules. Two typical Petri net models for station logic controllers are shown in Figure 17. Figure 17 (a) shows the hierarchical Petri net representation of an R-station. The normal operation cycle is represented by the directed circuit, $R_1t_1R_2t_2R_3t_4R_4t_5R_5t_6R_6t_8R_1$ and two reverse sequence are represented by the directed paths $R_3t_3R_2$ and $R_6t_7R_5$. The hierarchical Petri net representation of an I-station is shown in Figure 17 (b). The fault recovery sequences for the two I-modules are represented by the directed paths, $I_4t_5R_2t_2R_1$ and $I_5t_7R_2t_2R_1$. The directed circuit $R_1t_1R_2t_3R_3t_4I_4t_6I_5t_8R_6t_9R_1$ represents the normal operation cycle of the station.

From the hierarchical representations for station logic controllers, the Petri net structure of a

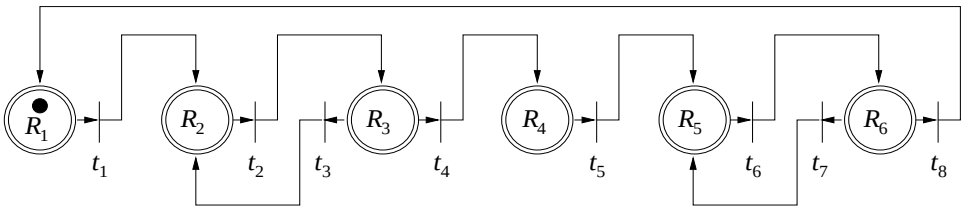


(a) R-module

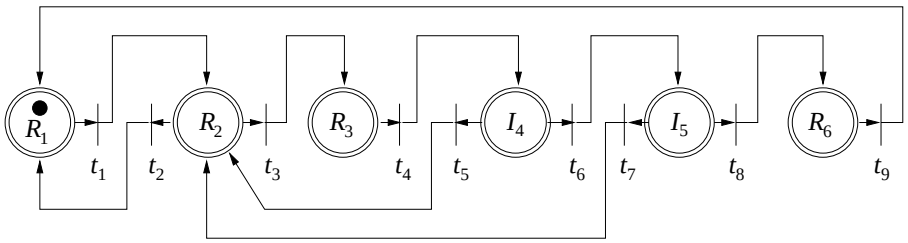


(b) I-module

Figure 16: The hierarchical representations for the operation modules



(a) The hierarchical Petri net representation of an R-station



(b) The hierarchical Petri net representation of a I-station

Figure 17: The hierarchical Petri net representations for station logic controllers

station logic controller can be characterized by the following proposition.

Proposition 1 *The Petri net representation of a station logic controller constructed according to the algorithm in Section 5.2.1 is a strongly connected state machine.*

Proof Let T_m be the set of all transitions within R- and T_c be I-modules and the set of transitions between modules. As shown in Figures 12 and 13, the Petri net structures of both the R-module and I-module are state machines, i.e. $|{}^\bullet t_i| = |t_i^\bullet| = 1$ for $t_i \in T_m$. Direct inspection shows that the R-module is a strongly connected state machine. The connection algorithm for operation sequences generates a simple directed path; only one place can be activated from the firing of each transition for operation module connection, i.e. $|t_i^\bullet| = 1$ for $t_i \in T_c$. Moreover, no synchronization exists in the operation sequences, i.e. $|{}^\bullet t_i| = 1$ for $t_i \in T_c$. Therefore, each transition in the Petri net representation of a station logic controller has one input place and one output place, i.e. $|{}^\bullet t_i| = |t_i^\bullet| = 1$ for $t_i \in T$ where $T = T_m \bigcup T_c$. Thus the Petri nets for station logic controllers are state machines. An R-station which consists of R-modules is strongly connected by the simple directed circuit for normal operation cycle and an I-station which consists of R- and I-modules is strongly connected by the simple directed circuit for normal operation cycle and the simple directed path(s) for fault recovery for irreversible operation(s). Therefore, the Petri net representation of a station logic controller is a strongly connected state machine. □

As shown in the Appendix, the concurrent operations in a station can be reduced to a place hierarchically. Therefore, the Petri net representation for the station logic controller with concurrent operations can be also considered a strongly connected state machine.

5.3 Internal Variables Re-set for Repetitive Cycles

Internal variable information is stored and updated within the operation modules. If a normal operation is completed, the value of the internal variable assigned for that operation is changed from 0 to 1; if its fault recovery operation is completed, the value is re-set from 1 to 0. This internal variable information can be utilized in other station logic controllers. For repetitive cyclic operation, all the internal variables in each station must be re-set from 1 to 0 whenever the cycle is finished. Unless this re-set is done properly, the operation causality between stations cannot be maintained correctly between cycles.

In our previous work [27], the initial state of a system was defined as the beginning moment of its timing bar chart. For each station, the moment when the first operation in the timing bar chart is initiated is used for the re-set of internal variables. In Petri nets, the input transition of the operating place ($p_O^{\mathcal{O}_{i,j}^N}$) in the first normal operation block of each station is considered the beginning of new cycle of the station; this transition is called *the initial transition*. The internal variables re-set control action for each station is assigned to the operating place ($p_O^{\mathcal{O}_{i,j}^N}$) of the normal operation block of the first operation listed in the timing bar chart. Note that this is not related to the home operation $\mathcal{O}_{1,j}$.

Internal variables are also used to encode the auto-start control logic. All the possible states in the normal operation cycle of a machining system are determined by the initial state; all reachable markings for the normal operation cycle, $R(N, M_O, \mathcal{NO})$ where $\mathcal{NO}$ represents the normal operation cycle, can be determined from the given initial marking for the normal operation cycle, (M_O,$\mathcal{NO}$) [27]. Because the home position for each station is chosen based on mechanical stability, the state for the home positions of all stations is normally not a reachable state of the normal operation cycle. Moreover, the state after fault recovery may not be a reachable state of the normal operation cycle. The control logic for auto-start from such states can be achieved without further considerations by the internal variable manipulation in a station logic controller. Because internal variables keep their information until the initial transition is fired, the auto mode can be re-started from any fault stop or the home position. On the other hand, the control logic for auto-stop can be achieved by assigning the input variable condition which implies the master stop is not activated to the output transition of the home place of each station. In other words, if the master stop is activated, the output transition of the home place cannot be fired, i.e. the station stops at its home place.

5.4 Mode Decision Control Logic

Operator commands are used to determine the control mode. The mode decision control logic takes these operator commands as inputs, and based on the state of the machining system determines the control mode. For example, a push button style input device is shown in Figure 18 (a). At the main console, five control inputs are considered: master-start, master-stop, auto, hand, and E-stop whereas at each station, the mode selection switch for station-auto and station-manual is considered. The master-start and master-stop push buttons generate operator commands for system initiation and system stop. After the station-auto is selected in all stations, the auto control mode of a system can be started by the auto push button in the main console. The control logic for the mode switchings shown in Figure 18 (b) depends on operator commands and fault signals. The initial marking represents the home position of a machining system.

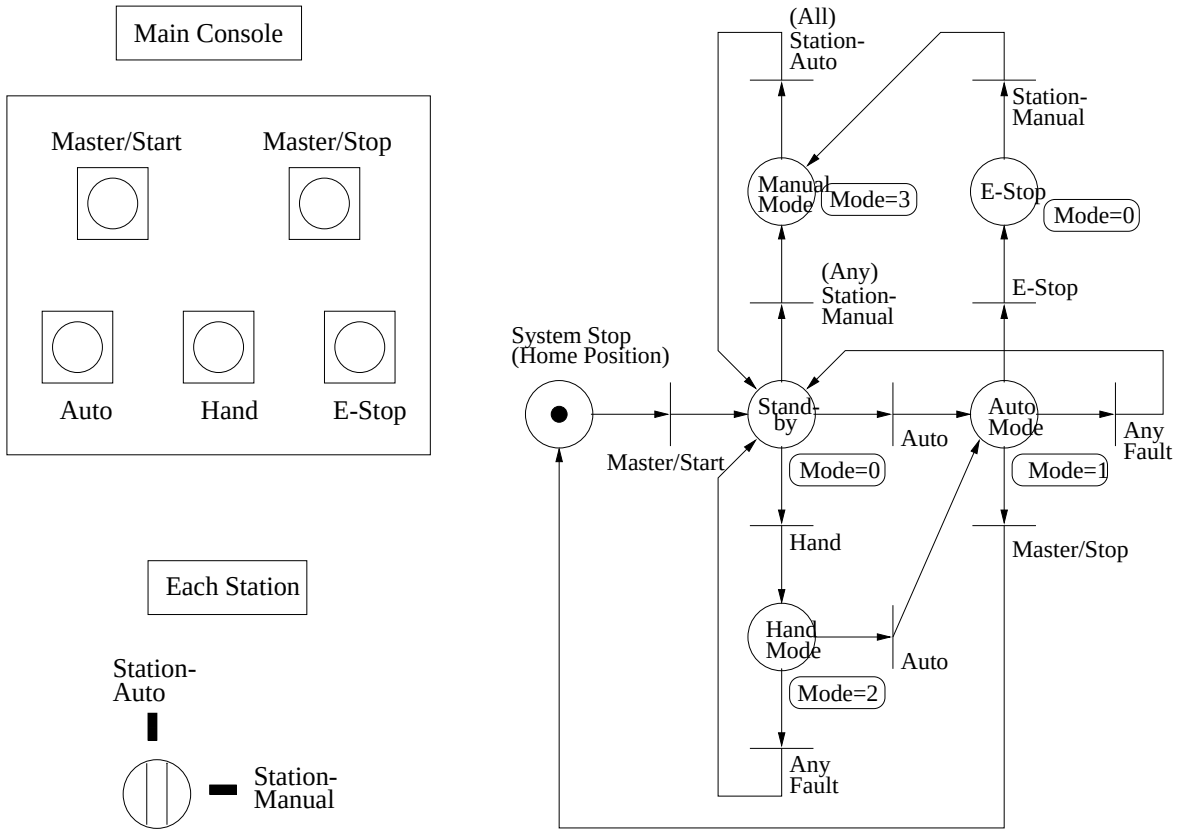
The mode decision control logic provides the station logic controllers information about the control mode of a system. Let "mode" in Figure 18 be an internal variable name for the system control mode; this internal variable information (the value of "mode") is used in the conditions of transitions of each station logic controller. The control logic for mode decision generates the value of "mode" with respect to operator commands or sensor signals. The control mode of a system is determined by the value of "mode". We assume that the values 1, 2, and 3 represent auto, hand, and manual mode respectively. If the token enters the "auto mode" place, control logic generates control action which is assigned to the place mode=1. If any fault occurs in the system at this state, the token moves to the "stand-by" place; control logic re-sets the value of mode to 0. In this stand-by state, the entire system is stopped and waits for an operator command deciding the next control mode. The transitions between the operation modules in the station logic controllers can be fired only in one of the three control modes. If the system stops by the E-stop operator command, the value of "mode" is also changed to 0; in this case, the system can only be recovered using the manual mode. From hand mode, the control mode can be changed to either the auto mode directly or to the stand-by state if a fault occurs.

5.5 Formal Analysis of Modular Logic Controller

The modular logic controller consists of Petri nets for mode decision control logic and station logic controllers. Each Petri net structure and its properties are considered first, then a condition for the modular logic controller to generate a correct control logic is investigated.

5.5.1 Properties of Each Control Module

By Proposition 1, we know that the control module for each station logic controller is a strongly connected state machine; its properties can be determined by the following well known theorems



(a) input devices for operator commands

(b) the Petri net representation

Figure 18: Control logic representation for control mode decision; the boxes represent the assigned control actions for places

of state machines.

Theorem 1 ([6]) *A state machine (N, M_0) is live if and only if N is strongly connected and M_0 has at least one token; a state machine (N, M_0) is safe if and only if M_0 has at most one token. A live state machine (N, M_0) is safe if and only if M_0 has exactly one token.*

Theorem 2 ([11]) *A live state machine is reversible.*

Therefore, the following proposition is straightforward to prove.

Proposition 2 *A station logic controller constructed according to the algorithm in Section 5.2.1 is live, safe, and reversible.*

Proof By Definition 7, the control module for each station logic controller in the modular logic controller has one token in the home position place. Therefore, it is a strongly connected state machine with a token in its initial marking, M_0. By Theorem 1 and 2, the control module for each station logic controller is live, safe, and reversible. □

In the Petri net model for mode decision control logic, as shown in Figure 18, the possible states of a machining system are represented by places: home position, stand-by, auto mode, hand mode, manual mode, and E-stop. Each state of the system is represented by a place and the initial marking is represented by a token in the home position place. The state of a machining system can be changed by an operator command or incoming sensor information (any fault); either of them is considered an input variable condition for a transition. The properties of this Petri net are determined by the following proposition.

Proposition 3 *The control module for mode decision control logic is a strongly connected state machine and is live, safe, and reversible.*

Proof Some states in the mode decision control logic can be changed to other states by the mutually exclusive conditions assigned to the output transitions: choice (decision or conflict) is allowed. The state of a machining system at any moment is represented by one of the possible states. Therefore, the input and output place of each transition must be always unique, $|{}^\bullet t| = |t^\bullet| = 1$; no synchronization is allowed. Moreover, all states of a system can be reached by operator commands or sensor events; there exists a directed path from every place to every other place. By the above reasoning, the control module for mode decision control logic is a strongly connected state machine which has one token in its initial marking. Moreover, it is live, safe, and reversible by Theorem 1 and 2. □

5.5.2 Operation Causality Condition

As mentioned in Section 3.1, a necessary condition for a Petri net representation to generate correct control logic for a logic controller is that its marked Petri net, $< N, M_0 >$, is live, safe, and reversible. In our formal representation, the control logic for a machining system was modeled by a set of Petri nets; each Petri net is live, safe, and reversible. However, the Petri nets for station logic controllers are implicitly connected by internal variable conditions to encode the operation dependencies between stations. Improper internal variable conditions can lead to deadlocks in station logic controllers (for example, operation B cannot start until operation A has finished, but A cannot start until B has finished). To prohibit these deadlocks, we introduce the *operation causality condition.*

In each control mode, operations are executed differently and the operation causality between stations is also different. Therefore, different internal variable conditions are necessary for each control mode. Because the control mode can change at any time, the internal variable conditions for all control modes must be considered simultaneously. Operation causality between stations is necessary for the following reasons.

Auto mode : prevent mechanical conflicts and reduce cycle time
Hand mode : coordinate multiple stations
Manual mode : prevent mechanical conflicts

Because some internal variable conditions in manual mode are replaced with other internal variable conditions in auto mode to reduce cycle time, and because the steps for the sequence of operator commands in hand mode are designed to satisfy the internal variable conditions for both auto mode and manual mode, only auto mode and manual mode internal variable conditions need to be considered for the operation causality condition. That is,

$$IVC_{occ} = IVC_{auto} \cup IVC_{manual}$$

where IVC_{auto} and IVC_{manual} are the set of internal variable conditions for auto and manual mode respectively and IVC_{occ} is the set of internal variable conditions which should be considered for the operation causality condition.

Within each station, the operations are totally ordered according to the timing bar chart. Combining the station operations with the operations whose internal variable conditions appear in that station results in a partially ordered set. This partial ordering is defined as the operation ordering for the station. The operation ordering for a station logic controller which has three normal operations and two internal variable conditions is illustrated in Figure 19; only the transitions in the normal operation cycle are shown for simplicity. In this station logic controller, there are two internal variable conditions, $IVC\ P$ and $IVC\ Q$, which represent the completion of operations P and

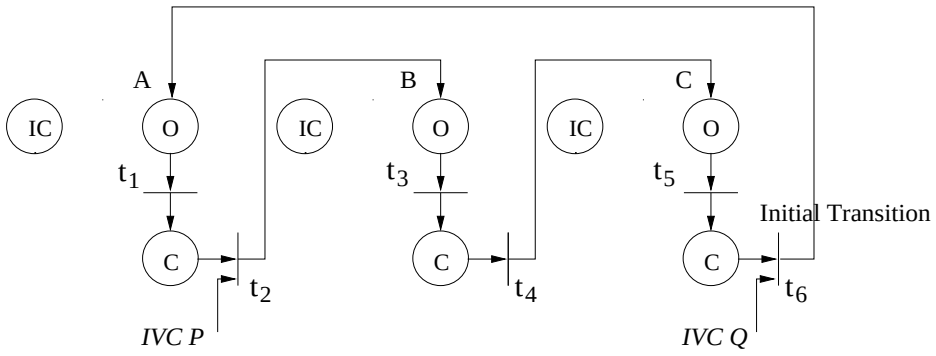


Figure 19: The operation ordering of a station logic controller: $\{A < B < C,\ P < B,\ Q < A\}$

Q in other stations. Then, the operation ordering for this station is $\{A < B < C,\ P < B,\ Q < A\}$. Using the operations ordering, we formalize the operation causality condition as follows.

Definition 8 (Operation Causality Condition) *A logic controller is said to satisfy the operation causality condition if the operations orderings in station logic controllers do not conflict with each other.*

The modular logic controller is a set of Petri nets which represents mode decision control logic and station logic controllers. Each Petri net is a strongly connected state machine; the liveness property is equivalent to the reversibility property (by Theorem 2). Using the operation causality condition, we can develop the following theorem for the properties of the modular logic controller.

Theorem 3 *The modular logic controller is live, safe and reversible if and only if each control module is live, safe, reversible, and the operation causality condition is satisfied.*

Proof (only if) Suppose that the modular logic controller is live, then all the Petri nets in the modular logic controller are live. If the control modules for station logic controllers do not satisfy the operation causality condition, they are deadlocked i.e. the Petri nets for station logic controllers are not live. This contradicts the assumption.
(if) If each control module is live, safe, and the operation causality condition is satisfied, then the Petri nets for station logic controllers and the Petri net for mode decision control logic are live and safe. This implies the modular logic controller is live and safe. Moreover, each Petri net is a strongly connected state machine; the liveness property is equivalent to the reversibility property (by Theorem 2). Therefore, if each control module is live, safe, reversible, and the control modules for station logic controllers satisfy the operation causality condition, the modular logic controller is live, safe and reversible. □

Since the liveness, safeness, and reversibility properties of the Petri net model for each station logic controller are guaranteed by following the connection algorithm in Section 5.2.1, the control

logic for a modular logic controller can be easily verified by checking the properties of the Petri net for the mode decision control logic and the operation causality condition.

6 Application

We have shown that the logic controller for a machining system can be represented by the modular logic controller which consists of live, safe, and reversible Petri nets. The correctness of its control logic can be verified by Theorem 3 using operation causality. Therefore, a logic controller can be constructed systematically using the modular logic controller representation. Moreover, due to its Petri net structure, the resulting modular logic controller can be easily implemented.

6.1 Automatic Code Generation

The machining systems which are considered in this paper, namely high volume transfer lines, have similar structures and functions. However, programming is still a time consuming job and always requires a "cycle and debug" stage to correct errors. This is due to the unstructured nature of the control logic and the lack of a verification method. If all or part of the logic can be generated automatically and the correctness of control logic can be verified before the mechanism has been built, the effects would be tremendous in the machining system industry. Our formal and modular representation of a logic controller can provide a way for this automatic code generation with verified control logic. The modular logic controller proposed in this paper has a formal Petri net structure and its correctness has been verified by the properties of Petri nets. Moreover, the control module for each station consists of operation modules. The control logic for each operation module is a function of input and output variables for the corresponding sensors and actuators; each operation module can be repeatedly generated by replacing the names of those variables.

6.1.1 Control Logic Generation from Operation Modules

The control program of a logic controller can be generated systematically using the Petri net representations for the operation modules. As mentioned in Section 2.5, a logic controller generates the control actions for a machining system with respect to sensor or operator command inputs. Input and internal variables are used for transition conditions, and output variables are used for control actions; all the inputs and outputs of a logic controller are assigned variable names. Internal variables are also assigned names. Therefore, each operation module can be considered a function of the corresponding variables. In other words, the control logic for similar operations can be repeatedly generated from the same Petri net structure by reassigning its variable names. The necessary variables for an R-module and their physical meanings are shown in Table 1.

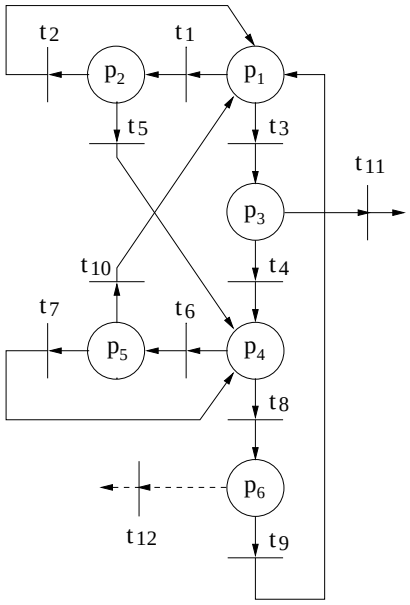
Table 1: The necessary variables and their physical meanings for an R-module

	Variable Name	Physical Meaning ($\_N$: normal operation, $\_F$: recovery operation)
Input	s_N, s_F	signals for operation completion
	f_N, f_F	signals for fault detection
	h_a, h_b, h_c	possible operator commands for normal sequence and reverse sequence in hand mode
	m_a, m_r	operator commands for manual mode (manual advance and manual return)
Output	a_N, a_F	actuators for operations
	d_N, d_F	display outputs for faults
Internal	IVC_N	operation completion

Table 2: The necessary variables and their physical meanings for an I-module

	Variable Name	Physical Meaning
Input	s_N	signal for operation completion
	f_N	signal for fault detection
	h_a	operator command for normal sequence in hand mode
	m_a, m_r	operator commands for manual mode (manual advance and manual return)
Output	a_N	actuator for the operation
	d_N	display output for faults
Internal	IVC_N	operation completion

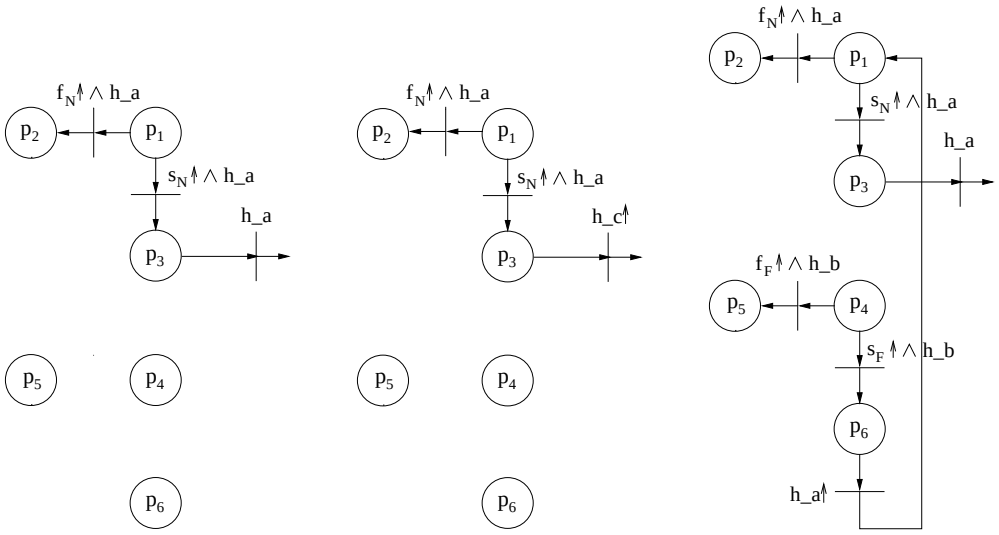
The control logic for an R-module is shown in Figure 20; it depends on the operation sequence assigned to each operator command. For example, some reversible operations do not require a reverse sequence in hand mode; the fault recovery operation is sufficient. In this case, the transition t_{12} is not necessary in manual and hand modes. The possible transition conditions in the hand mode control logic for an R-module are illustrated in Figure 21; each case is considered a *hand mode type*. Because the types of manual mode can be determined by the hand mode type, the control logic for each R-module can be generated by assigning its variables to the appropriate hand mode type. The brackets in the first table in Figure 20 represent conditions which may occur in the different hand mode types. Because no condition is necessary for the transition t_{11} in auto mode, it is represented by "N.C.". Table 2 and Figure 22 show the necessary variables and the control logic for an I-module, respectively. Typically, the irreversible operations in a machining station are initiated by one operator command in hand mode such as "cycle all machining stations."



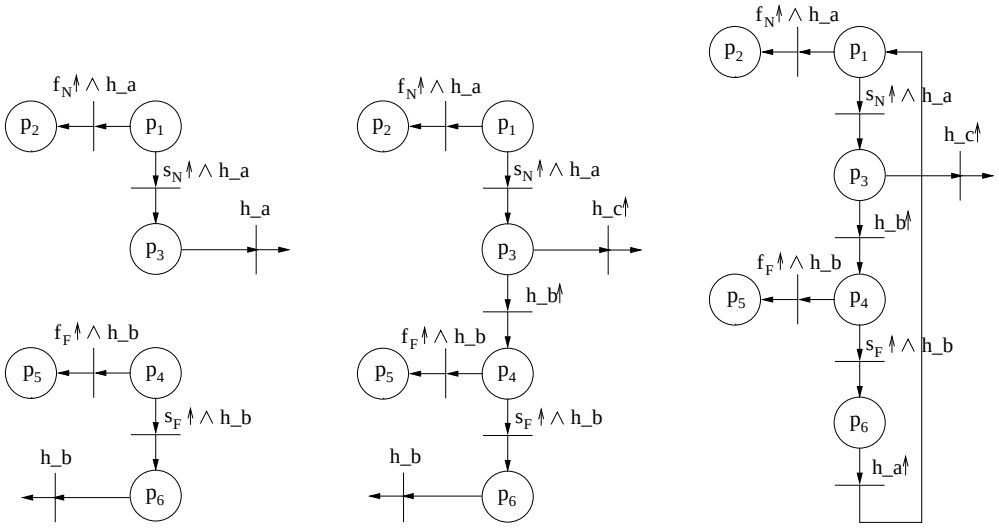
Transitions	Conditions		
	auto mode	hand mode	manual mode
t_1	$f_N\uparrow$	$f_N\uparrow \wedge h\_a$	$m\_a\downarrow$
t_2			$m\_a\uparrow$
t_3	$s_N\uparrow$	$s_N\uparrow \wedge h\_a$	$s_N\uparrow \wedge (m\_a \vee m\_r)$
t_4		$[h\_b\uparrow]$	$m\_r\uparrow$
t_5			$m\_r\uparrow$
t_6		$[f_F\uparrow \wedge h\_b]$	$m\_r\downarrow$
t_7			$m\_r\uparrow$
t_8		$[s_F\uparrow \wedge h\_b]$	$s_F\uparrow \wedge (m\_r \vee m\_a)$
t_9		$[h\_a\uparrow]$	$m\_a\uparrow$
t_{10}			$m\_a\uparrow$
t_{11}	N.C.	$h\_a\ [h\_c\uparrow]$	$m\_a\uparrow$
t_{12}		$[h\_b]$	$[m\_r\uparrow]$

places	control actions	places	control actions
p_1	$a_N = 1$	p_4	$a_F = 1$
p_2	$a_N = 0$ & $d_N = 1$	p_5	$a_F = 0$ & $d_F = 1$
p_3	$a_N = 0$ & IVC_N=1	p_6	$a_F = 0$ & IVC_N=0

Figure 20: The control logic for an R-module

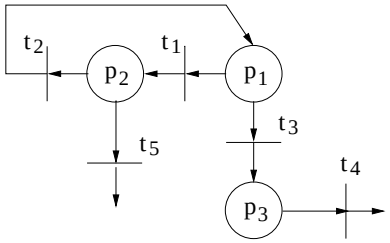


(a) Type 1 (b) Type 2 (c) Type 3



(d) Type 4 (e) Type 5 (f) Type 6

Figure 21: The transition conditions for each hand mode type. Type 1 is used for the first and intermediate operations in a sequence of normal operations (no reverse sequence); Type 2 is for the last operation in a sequence of normal operations (no reverse sequence); Type 3 is for the first operation in a sequence of normal operation with a reverse sequence; Type 4 is for intermediate operations in a sequence of normal operation with a reverse sequence; Type 5 is for the last operation in a sequence of normal operation with a reverse sequence; Type 6 is for an operation which is started by one operator command and reversed by another. The variables *h_a*, *h_b*, and *h_c* indicate the possible operator commands for normal and reverse sequences.



Transitions	Conditions		
	auto mode	hand mode	manual mode
t_1	$f_N\uparrow$	$f_N\uparrow \wedge h\_a$	$m\_a\downarrow$
t_2			$m\_a\uparrow$
t_3	$s_N\uparrow$	$s_N\uparrow \wedge h\_a$	$s_N\uparrow \wedge (m\_a \vee m\_r)$
t_4	N.C.	h_a	$m\_a\uparrow$
t_5			$m\_r\uparrow$

places	control actions
p_1	$a_N = 1$
p_2	$a_N = 0$ & $d_N = 1$
p_3	$a_N = 0$ & IVC_N=1

Figure 22: The control logic for an I-module

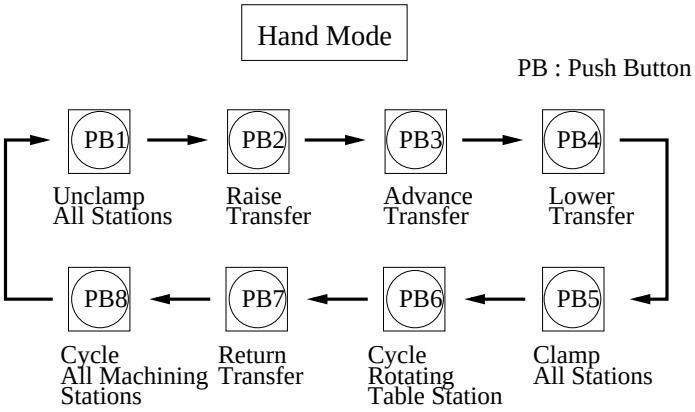


Figure 23: The input device for hand mode

6.1.2 Illustrative Example

As a simple illustrative example, the control logic for three stations from the transfer line in Figure 2 will be considered here. The three stations are the transfer bar, clamp 1, and mill 1. The operator commands for hand mode will be needed in the control logic; the input device for the whole system hand mode is shown in Figure 23.

The ordered sequences of operations for three stations are obtained from the timing bar chart in Figure 2. The transfer bar, clamp 1, and mill 1 stations are named as stations 1, 2, and 3 respectively. Then the ordered sequence for each station can be represented by the following equations.

$$\mathcal{S}_1 = \mathcal{O}_{1,1}, \mathcal{O}_{1,2}, \mathcal{O}_{1,3}, \mathcal{O}_{1,4}, \mathcal{O}_{1,5}, \mathcal{O}_{1,6}, \mathcal{O}_{1,7}, \mathcal{O}_{1,8}, \mathcal{O}_{1,9}, \mathcal{O}_{1,10}$$

$$\mathcal{S}_2 = \mathcal{O}_{2,1}, \mathcal{O}_{2,2}$$

$$\mathcal{S}_3 = \mathcal{O}_{3,1}, \mathcal{O}_{3,2}, \mathcal{O}_{3,3}, \mathcal{O}_{3,4}, \mathcal{O}_{3,5}$$

The home operations ($\mathcal{O}_{1,j}$'s) for the station are "return transfer", "return clamp", and "reset main slide" respectively and the details for each operation are shown in Table 3. The hand mode type in the table is determined by the operation sequence specifications for hand mode. The operator commands in hand mode are assigned to the input variables representing their push buttons PB_i, i= 1 to 8. The normal operation sequences, $\mathcal{O}_{1,2}, \mathcal{O}_{1,3}, \mathcal{O}_{1,4}, \mathcal{O}_{1,5}$ and $\mathcal{O}_{1,7}, \mathcal{O}_{1,8}, \mathcal{O}_{1,9}, \mathcal{O}_{1,10}$ need reverse sequences for the repeatable steps in hand mode: "raise transfer" and "lower transfer". The irreversible operation in the mill 1 station is represented by an I-module and its fault recovery can be achieved by the fault recovery operation of the "rapid advance positioning slide" operation ("rapid return positioning slide") and the "reset main slide" operation.

The control logic for three stations is completed by assigning the internal variable conditions for each control mode. The hierarchical representation for the control logic of three stations are shown

Table 3: The details of operations in the system

Operations	Operation Module	Hand Mode Type & Input Variables (h_a, h_b, h_c)
Return Transfer ($\mathcal{O}_{1,1}$)	$R_{1,1}$	type 6 (PB_7, PB_3, PB_2)
1st Lift ($\mathcal{O}_{2,1}$)	$R_{2,1}$	type 3 (PB_2, PB_4, —)
2nd Lift ($\mathcal{O}_{3,1}$)	$R_{3,1}$	type 4 (PB_2, PB_4, —)
3rd Lift ($\mathcal{O}_{4,1}$)	$R_{4,1}$	type 4 (PB_2, PB_4, —)
4th Lift ($\mathcal{O}_{5,1}$)	$R_{5,1}$	type 5 (PB_2, PB_4, PB_3)
Advance Transfer ($\mathcal{O}_{6,1}$)	$R_{6,1}$	type 6 (PB_3, PB_7, PB_4)
1st Lower ($\mathcal{O}_{7,1}$)	$R_{7,1}$	type 3 (PB_4, PB_2, —)
2nd Lower ($\mathcal{O}_{8,1}$)	$R_{8,1}$	type 4 (PB_4, PB_2, —)
3rd Lower ($\mathcal{O}_{9,1}$)	$R_{9,1}$	type 4 (PB_4, PB_2, —)
4th Lower ($\mathcal{O}_{10,1}$)	$R_{10,1}$	type 5 (PB_4, PB_2, PB_7)
Return Clamp ($\mathcal{O}_{1,2}$)	$R_{1,2}$	type 6 (PB_3, PB_7, PB_7)
Advance Clamp ($\mathcal{O}_{2,2}$)	$R_{2,2}$	type 6 (PB_7, PB_3, PB_3)
Reset Main Slide ($\mathcal{O}_{1,3}$)	$R_{1,3}$	type 2 (PB_8, —, PB_8)
Rapid Adv. Pos. Slide ($\mathcal{O}_{2,3}$)	$R_{2,3}$	type 1 (PB_8, —, —)
Decel ($\mathcal{O}_{3,3}$)	$R_{3,3}$	type 1 (PB_8, —, —)
Feed Main Slide ($\mathcal{O}_{4,3}$)	$I_{4,3}$	type 1 (PB_8, —, —)
Rapid Ret. Pos. Slide ($\mathcal{O}_{5,3}$)	$R_{5,3}$	type 1 (PB_8, —, —)

in Figure 24 with the internal variable conditions for auto mode.

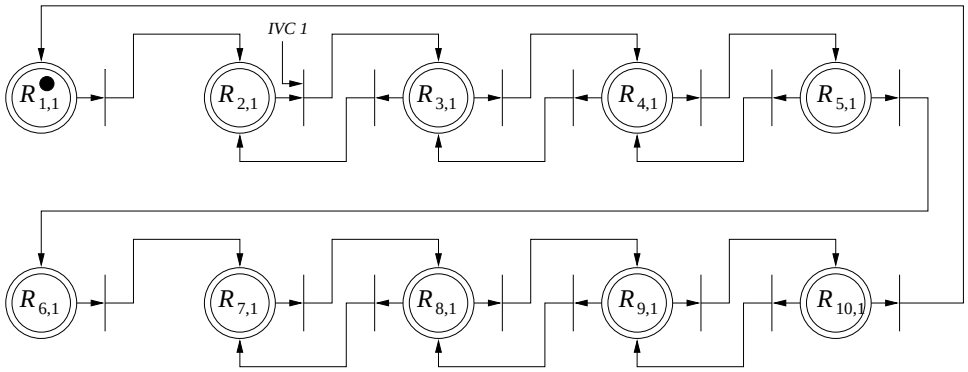
6.2 Implementation

Once a logic controller has been represented by Petri nets in the form of a modular logic controller, and verified using Theorem 3, the next step is to implement the logic controller.

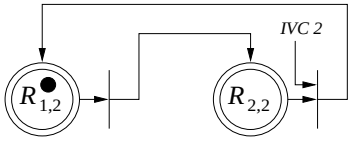
6.2.1 Implementation by SFC

Currently, PLCs are the most popular devices for implementing logic controllers. As an international standard, the IEC 1131-3 programming languages were developed for PLCs [38]. Since the first revision of the IEC 1131-3 standard published in 1993, the PLCs from major PLC manufacturers can be programmed using some of the programming languages provided from IEC 1131-3 standard. One of the IEC 1131-3 standard languages is SFC (Sequential Function Chart). It is based on Grafcet which was inspired from Petri nets and in fact, the structures of SFC and Grafcet are almost the same. The comparison of Grafcet and Petri nets along with SFC can be found in [30].

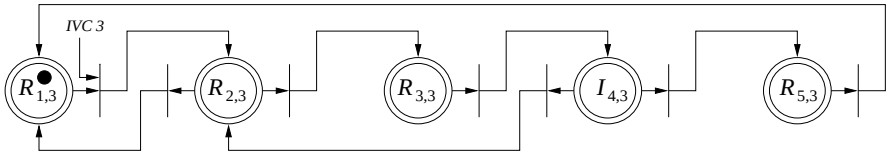
A live, safe, and reversible Petri net which has mutually exclusive transitions can be implemented directly in SFC [30]. The modular logic controller developed in this paper consists of live, safe, and



(a) Transfer bar station control logic



(b) Clamp 1 station control logic



(c) Mill 1 station control logic

Figure 24: The control logic for three stations with their internal variable conditions for auto mode (IVC 1: $\mathcal{O}_{1,2}$=1, IVC 2: $\mathcal{O}_{10,1}$=1, IVC 3: $\mathcal{O}_{2,2}$=1)

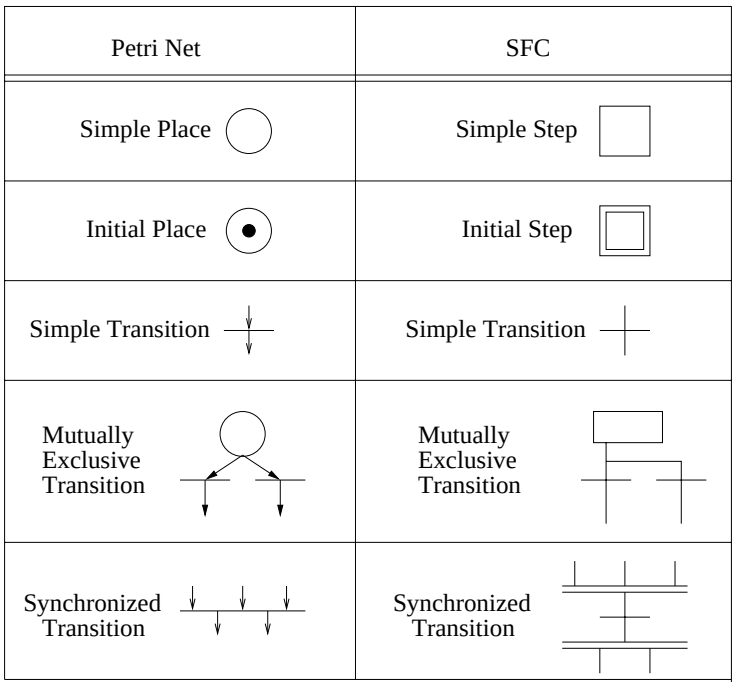


Figure 25: The conversion rules between Petri net and SFC

reversible Petri nets with mutually exclusive transitions. Therefore, it can be directly implemented in SFC; the graphical conversion rule is shown in Figure 25.

6.2.2 Other Implementations

Petri net-based control logic can be also implemented directly by software or hardware using microprocessor-based controllers; the implementation using high-level concurrent programming languages was introduced in [39, 40] and hardware-based implementation was considered in [41, 42, 43]. In fact, Petri net-based logic controllers have been applied directly to some manufacturing systems [44, 45]. However, these Petri net based logic controllers were not employed broadly in industry. Because the Petri net formalisms were modified to fit their specific manufacturing systems, the methodologies were difficult to implement in existing PLC's and they lacked verification in the design stage. Perhaps more importantly, however, the logic controllers were developed only for the normal operation cycle. As mentioned in Section 2, exception handling is important in logic controllers, especially for machining systems, and may represent 90% of the required control logic.

In this paper, we developed a modular Petri net structure for station logic controllers which includes the specifications for fault stop/fault diagnosis and fault recovery. Each modular structure is simple and does not require any special Petri net formalism. Therefore, it can easily be implemented in PLC's using SFC and other PLC programming languages. Any specifications can be

added using the superposition design methodology. Moreover, the correctness of the control logic can be guaranteed by the structure of the modular station logic controller.

6.3 Reconfigurable Logic Controllers

High volume transfer lines have traditionally been designed as dedicated manufacturing systems. However, they need a certain degree of reconfiguration due to shorter product life cycles and increasing product variety in the automotive industry. Modularity is one of the key characteristics of reconfigurability [1, 2]. In fact, modular standards have been developed by machining system builders for maximal re-use of machine components. Each component is designed to be easily and quickly changed to reduce machine set-up and ramp-up time when reconfiguration is required. A modular design methodology for machine tools was introduced in [37]. However, the modular design concept for the control logic is not well-developed enough to realize reconfigurable machining systems. In fact, decentralized logic controllers are used in industry as modular logic controllers. In a decentralized logic controller, a PLC is assigned to a station or stations (normally two or three depending on the complexity of the mechanical stations). These PLC's communicate with each other over a network. However, we have found that while the hardware is modularized, the control logic is not.

In [28], a modular logic controller for the normal operation cycle was introduced as a causal Petri net representation of a timing bar chart. The control logic for each station was considered as a module. With this modular structure, a logic controller in the normal operation cycle can be reconfigured efficiently when some stations in the system need to be replaced with new ones or new stations are added to the system. In this paper, a modular logic controller is developed for other specifications such as exception handling, and two different modular structures are introduced for operations. With this type of modular structure, any reconfiguration of the operations in a station can be achieved with maximum re-use of the old algorithm by changing the control actions and the transition conditions. Any level of reconfiguration in a logic controller can be achieved using these modular structures.

7 Conclusion

In this paper, a new representation method for a logic controller with three control modes is introduced using the Petri net formalism. Two operation modules are developed with respect to the characteristics of fault recovery processes of operations in machining systems. The connection algorithm for operation modules to represent various operation sequences in station logic controllers are provided. Using internal variable conditions and superposition, the control logic with three control modes can be simply represented. The control logic for a machining system is represented

by a modular logic controller which consists of a control module for mode decision and control modules for station logic controllers; we have shown that each control module is represented by a live, safe, and reversible Petri net. A condition for the modular logic controller to generate a correct control logic is considered. In our formal representation, the control logic is modularized. A logic controller can be easily reconfigured and its control logic can be generated automatically by its modular structure. Moreover, our modeling methodology starts from the basic control unit for an operation, which can be applied to other types of manufacturing system controllers.

A Connection Rules for Concurrent Operations

Frequently, a machining station has some concurrent operations in its operations sequence. If we only consider the normal operation cycle, these concurrent operations can be simply represented by *parallel places* [27]. However, if the fault recovery process with two operator commands (advance and return) in the manual control mode is considered, the operation modules should be connected in such a way that:

- if a reverse sequence is necessary for the concurrent operations, the operations should be reversed concurrently.
- each concurrent operation can be recovered independently from its fault stop.

We assume that the concurrent operations are only used with reversible operations because irreversible operations in machining stations are typically executed consecutively. Therefore, concurrent operations are represented by R-modules. The connection for two concurrent operations is shown in Figure 26. Concurrency in the normal sequence is achieved by the transitions t_1 and t_2; concurrency in the reverse sequence can be achieved by the transitions t_3 and t_4. The four internal variable conditions ($IVC\ A$, $IVC\ A'$, $IVC\ B$, and $IVC\ B'$) allow the fault recovery functions to be executed independently. The initial marking in the figure represents the state in which the operation A is in fault stop and the operation B is completed. Because of the internal variable condition $IVC\ A$, only the transition t_r is activated by the token, allowing the fault in A to be recovered independently of B.

Because concurrent operations are executed the same way in their normal and reverse sequence, all the operation modules for concurrent operations can be represented by a single place. This hierarchical structure is illustrated in Figure 27.

Acknowledgments. This research was supported by the NSF Engineering Research Center for Reconfigurable Machining Systems based at the University of Michigan, Ann Arbor, under grant number EEC9529125. We would like to thank Matthew C. VanGilder, Michael R. Griffin, Barry

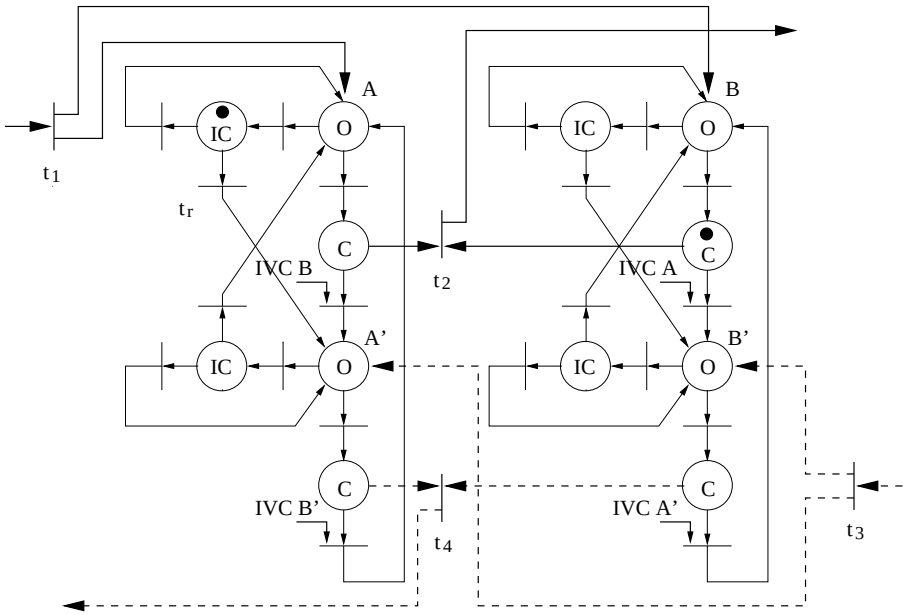


Figure 26: The connection of two concurrent operations

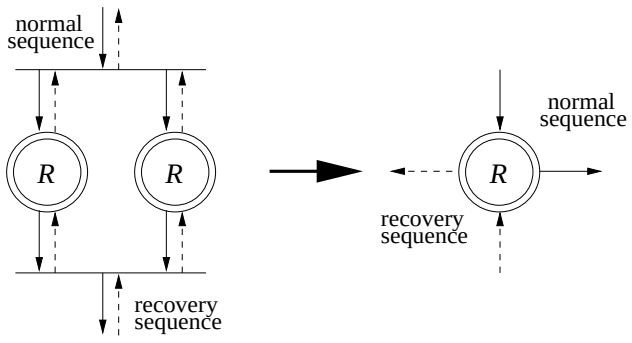


Figure 27: The hierarchical representation for concurrent operations

Gardner, and Bryan A. Graham of Lamb Technicon for providing us with the example used in this paper, and for their contributions to this research project.

References

[1] Y. Koren and A. G. Ulsoy, "A Science-Base for Reconfigurable Manufacturing Systems," Technical Report 1, ERC/RMS, 1997.

[2] M. Mehrabi, A. Ulsoy, and Y. Koren, "Reconfigurable Manufacturing Systems: Key to Future Manufacturing," in *The 1998 Japan-USA Symposium on Flexible Automation*, pp. 677–682, Otsu, Japan, July 1998.

[3] J. Bigou and *et al*, "A Methodology of Specification and Implementation of Distributed Discrete Control Systems," *IEEE Transactions on Industrial Electronics*, vol. IE-34, no. 4, pp. 417–421, November 1987.

[4] R. H. Weston, R. Harrison, A. H. Booth, and P. R. Moore, "Universal Machine Control System Primitives for Modular Distributed Manipulator Systems," *International Journal of Production Research*, vol. 27, no. 3, pp. 395–410, 1989.

[5] P. Kopacek, G. Kronreif, and R. Probst, "A Modular Control System for Flexible Robotized Manufacturing Cells," *Robotica*, vol. 17, pp. 23–32, 1999.

[6] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 5, pp. 541–580, April 1989.

[7] R. David and H. Alla, "Petri Nets for Modeling of Dynamic Systems - A Survey," *Automatica*, vol. 30, no. 2, pp. 175–202, 1994.

[8] R. Zurawski and M. Zhou, "Petri Nets and Industrial Applications: A Tutorial," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 6, pp. 567–582, December 1994.

[9] L. E. Holloway, B. H. Krogh, and A. Giua, "A Survey of Petri Net Methods for Controlled Discrete Event Systems," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 7, pp. 151–190, 1997.

[10] K.-P. Brand and J. Kopainsky, "Principles and Engineering of Process Control with Petri Nets," *IEEE Transactions on Automatic Control*, vol. 33, no. 2, pp. 138–149, February 1988.

[11] F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat, *Practice of Petri nets in Manufacturing*, Chapman & Hall, New York, first edition, 1993.

[12] M. C. Zhou, F. DiCesare, and D. L. Rudolph, "Design and Implementation of a Petri Net Based Supervisor for a Flexible Manufacturing System," *Automatica*, vol. 28, no. 6, pp. 1199–1208, 1992.

[13] M. Zhou, F. DiCesare, and A. A. Desrochers, "A Hybrid Methodology for Synthesis of Petri Net Models for Manufacturing Systems," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 350–361, June 1992.

[14] L. Ferrarini, "Hierarchical Multi-layer CAD System for Systematic Design of Complex Logic Controllers," *Robotics & Computer-Integrated Manufacturing*, vol. 12, no. 2, pp. 173–184, 1996.

[15] L. Ferrarini, M. Narduzzi, and M. Tassan-Solet, "A New Approach to Modular Liveness Analysis Conceived for Large Logic Controllers' Design," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 2, pp. 169–184, April 1994.

[16] L. Ferrarini, "An Incremental Approach to Logic Controller Design with Petri Nets," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 3, pp. 461–473, May 1992.

[17] J. Zaytoon, "Specification and Design of Logic Controllers for Automated Manufacturing Systems," *Robotics & Computer-Integrated Manufacturing*, vol. 12, no. 4, pp. 353–366, 1996.

[18] A. Giua and F. DiCesare, "Petri Net Structural Analysis for Supervisory Control," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 2, pp. 185–195, April 1994.

[19] L. Ferrarini and C. Maffezzoni, "Designing Logic Controllers with Petri Nets," in *IFAC Computer Aided Design in Control Systems*, pp. 319–324, Swansea, UK, 1991.

[20] K. Venkatesh, M. Kaighobadi, M. Zhou, and R. J. Caudill, "Augmented Timed Petri Nets for Modeling, Simulation, and Analysis of Robotic Systems with Breakdowns," *Journal of Manufacturing Systems*, vol. 13, no. 4, pp. 289–301, 1994.

[21] M. C. Zhou and F. DiCesare, "Adaptive Design of Petri Net Controllers for Error Recovery in Automated Manufacturing Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 963–972, September 1989.

[22] M. D. Jeng, "Petri Nets for Modeling Automated Manufacturing Systems with Error Recovery," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 5, pp. 752–760, October 1997.

[23] N. Viswanadham and T. L. Johnson, "Fault Detection and Diagnosis of Automated Manufacturing Systems," in *Proceedings of the 27th Conference on Decision and Control*, pp. 2301–2305, Austin, Texas, 1998.

[24] Y. Koren, "Cross-Coupled Biaxial Computer Control for Manufacturing Systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, pp. 265–272, December 1980.

[25] E. J. Davison and I. J. Ferguson, "The Design of Controllers for the Multivariable Robust Servomechanism Problem Using Parameter Optimization Methods," *IEEE Transactions on Automatic Control*, vol. AC-26, no. 1, pp. 93–110, December 1981.

[26] T. P. McDunn, "The Simple Approach to Transfer Line Control," in *IPC '92: Enabling Flexibility*, pp. 297–306, 1992.

[27] E. Park, D. M. Tilbury, and P. P. Khargonekar, "Modular Logic Controller for Machining Systems: Formal Representation and Performance Analysis using Petri Nets," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 6, pp. 1046–1061, December 1999.

[28] E. Park, D. M. Tilbury, and P. P. Khargonekar, "A Formal Implementation of Logic Controllers for Machining Systems using Petri Nets and Sequential Function Chart," in *Proceedings of The 1998 Japan-USA Symposium on Flexible Automation*, pp. 683–690, Otsu, Japan, July 1998.

[29] W. Reisig, *Petri Nets*, Springer-Verlag, Berlin, 1982.

[30] R. David, "Grafcet: A Powerful Tool for Specification of Logic Controllers," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 3, pp. 253–268, September 1995.

[31] P. Freedman and A. Malowany, "The Analysis and Optimization of Repetition within Robot Workcell Sequencing Problems," in *Proc. IEEE International Conf. on Robotics and Automation*, pp. 1276–1281, 1988.

[32] B. Krogh and R. Sreenivas, "Essentially Decision Free Petri Nets for Real-Time Resource Allocation," in *Proc. IEEE International Conf. on Robotics and Automation*, pp. 1005–1011, 1987.

[33] X.-R. Cao and Y.-C. Ho, "Models of Discrete Event Dynamic Systems," *IEEE Control Systems Magazine*, vol. 10, no. 4, pp. 69–76, June 1990.

[34] T. L. Booth, *Sequential Machines and Automata Theory*, John Wiley and Sons, Inc., New York, 1967.

[35] T. Murata and J. Y. Koh, "Reduction and Expansion of Live and Safe Marked Graphs," *IEEE transactions on circuits and systems*, vol. CAS-27, no. 1, pp. 68–70, January 1980.

[36] Y. Koren, "A Science-Base for Reconfigurable Machining Systems Vision with Examples," Technical Report 19, ERC/RMS, 1999.

[37] Y.-M. Moon and S. Kota, "Generalized Kinematic Modeling Method for Reconfigurable Machine Tools," in *25th Biannual mechanism design conference*, p. MECH/5946, Atlanta, GA, 1998.

[38] R. W. Lewis, *Programming Industrial Control Systems using IEC 1131-3*, The Institution of Electrical Engineers, London, United Kingdom, 1995.

[39] J. Colom, M. Silva, and J. Villarroel, "On Software Implementation of Petri Nets and Colored Petri Nets using High-Level Concurrent Languages," in *Proc. of the European Workshop and Application and Theory of Petri Nets*, 1986.

[40] M. Silva and S. Velilla, "Programmable Logic Controllers and Petri Nets: A Comparative study," in *Proceedings of the third IFAC/IFIP Symposium*, pp. 83–88, Madrid, Spain, 1982.

[41] M. Auguin, F. Boeri, and C. Andre, "Systematic Method of Realization of Interpreted Petri Nets," *Digital Processes*, vol. 6, no. 1, pp. 55–68, 1980.

[42] F. Anzai and *et al.*, "Hardware Implementation of a Multiprocessor System Controlled by Petri Nets," in *Proceedings of IECON'93*, pp. 121–126, 1993.

[43] N. Chang, W. H. Kwon, and J. Park, "Hardware Implementation of Real-Time Petri Net-Based Controllers," in *The 4th IFAC Workshop on Algorithm and Architecture for Real-time Control*, 1997.

[44] T. Murata, O. Komoda, K. Matsumoto, and K. Haruna, "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation," *IEEE Transactions on Industrial Electronics*, vol. IE-33, no. 1, pp. 1–8, February 1986.

[45] D. Crockett, A. Desrochers, F. DiCesare, and T. Ward, "Implementation of a Petri Net Controller for a Machining Workstation," in *Proc. IEEE International Conf. on Robotics and Automation*, pp. 1861–1867, 1987.