View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Lehigh University: Lehigh Preserve

Lehigh University
Lehigh Preserve

Theses and Dissertations

2003

Control of the Lehigh University flexible manufacturing cell using sensor-based stage Petri net modeling

Michael J. Smith
Lehigh University

Follow this and additional works at: http://preserve.lehigh.edu/etd

Recommended Citation
Smith, Michael J., "Control of the Lehigh University flexible manufacturing cell using sensor-based stage Petri net modeling" (2003). *Theses and Dissertations.* Paper 805.

This Thesis is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

Smith, Michael J.

Control of the Lehigh University Flexible Manufacturing Cell Using Sensor-Based Stage...

May 2003

Control of the Lehigh University Flexible Manufacturing Cell Using Sensor-Based Stage Petri Net Modeling

By

Michael J. Smith

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

May 2003

Certificate of Approval

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

4/24/03
Date

Thesis Advisor

Chairperson of Department

Acknowledgements

I would like to thank Prof. Nicholas Odrey for all of his help and guidance on this thesis. He has been of great inspiration and help during both my graduate and undergraduate career at Lehigh University. I would also like to thank Hyung Kim, the laboratory technician in the Robotics Lab.

Table of Contents

Abstract 1

Chapter 1 – Introduction 2

Chapter 2 – Basic Concepts of Petri Nets 5

2.1 Introduction 5

2.2 Types of Petri Nets 7

2.3 Properties 9

Chapter 3 – Sensor- Based Stage Petri Nets 12

3.1 Introduction 12

3.2 Stage Programming of Discrete-Event Control Systems 15

3.3 Sensor-Based Stage Petri Net 17

3.4 Stage Control Net 19

3.5 Input Contacts 21

3.6 Summary 22

Chapter 4 – Sensor-Based Stage Petri Net Modeling and the Lehigh University Flexible Manufacturing Cell 23

4.1 System and Scenarios 23

4.2 Inputs and Outputs 25

4.3 Logic Combinations and Boolean Equations 28

4.4 Integration of Stages 34

Chapter 5 – Conclusions 37

Chapter 6 – Recommendation for Future Work 38

References 39

Appendix A 41

Appendix B 47

Vita 56

List of Figures

Figure 1 – Example of Basic Petri Net 6

Figure 2 – Ladder Logic Diagram Example 12

Figure 3 – Types of Contacts 21

Figure 4 – FMS Layout 23

Figure 5 – Cell Layout with Sensor Positions 26

Figure 6 – SCN U RTPN for Equation (2) 31

Figure 7 – SBSPN for Entire System 36

Abstract

Automated manufacturing systems have been around for several decades, and the use of a programmable logic controller has been one of the most popular ways to control these systems. When use of PLCs first began, ladder logic diagrams were used to program the logic into the PLC. As systems have become more complex, however, the use of Petri nets has been of great interest for programming PLCs. The concept of using a sensor-based stage Petri net has emerged as a simplified way in which to program a PLC for a complex system, and it is this particular method that is the topic of the approach taken in this thesis. Specifically, the following topics were covered:

(1) Explain the general theory behind Petri nets, and their application to manufacturing,

(2) Introduce the theory of sensor-based stage Petri nets,

(3) Describe the equipment contained within the Lehigh University flexible manufacturing cell, and how sensor-based stage Petri nets can be developed.

The constructed nets were investigated for two different scenarios. Each scenario indicated the benefits of sensor-based stage Petri nets in a manufacturing environment.

Chapter One – Introduction

In today's modern world of manufacturing, there is a greater emphasis being placed on the use or robotics and automation in a production facility. One area of particular interest is the control of these individual machines as well as the control of the system as a whole. The most common method of system and machine control is the use of a programmable logic controller, or PLC.

PLCs were first introduced in the late 1960s with the intent of replacing mechanical switching modules [Chang, 1998]. Modern PLCs have had their capabilities extended greatly from the earlier versions, and today are capable of performing many functions. These include digital processing and high-speed communication, computer language support, and process control.

The manner in which a PLC is programmed originally stemmed from the relays they were used to replace. The people who programmed them used a language similar to that of the electrical technicians who designed circuits: ladder logic diagrams. These diagrams were sufficient for simple systems, but as the complexity of the system grew, the downfalls of the ladder logic diagrams became apparent. Writing ladder logic diagrams for large, complex systems and debugging them proved to be very difficult [Chang, 1998].

The 1960s also saw the development of Petri nets, which turned into a powerful mathematical and graphical modeling tool. As ladder-logic diagrams became more and more difficult to implement as a means of programming PLCs, the use of Petri nets emerged as a possible alternative to model increasingly complex system [David, 1992]. These complex systems could be easily modeled with Petri nets, then the nets could be turned back into the ladder logic to program the PLC. Use of Petri nets also allowed the user to evaluate system properties much easier than other methods.

Here, the focus is on the use of Petri nets for modeling and control of the Lehigh University flexible manufacturing cell by applying a sensor-based stage approach. When Petri nets are used to model and control a manufacturing system, the common meaning of a place is a resource that is needed or released, and a transition is a physical task or job. The sensor-based stage approach uses inputs and outputs from sensors placed throughout the system to control the discrete event actions of the system.

The work here is presented in the following manner: Chapter Two explains the basic theories and concepts of Petri nets; Chapter Three presents the theory behind sensor-based stage Petri net modeling; Chapter Four discusses the application of sensor-based stage Petri net modeling to the flexible manufacturing cell at Lehigh;

Chapter Five presents the conclusions reached from this procedure; and Chapter Six presents possible future work and modifications to the system.

Chapter Two – Basic Concepts of Petri Nets

2.1 Introduction

Petri nets were originally developed in the 1960s by C.A. Petri as a mathematical modeling tool for distributed systems. Petri nets are capable of serving a dual purpose: they can be both a mathematical tool as well as a graphical tool [Moore, 1995]. When used as a mathematical tool, Petri nets can be used to develop state equations, algebraic equations, and various other mathematical models. When used as a graphical tool, Petri nets allow the user to create a visual model of a dynamic system.

A simple Petri net is defined by four basic elements: places, tokens, transitions, and arcs. Graphically, the places are represented by circles, tokens are represented by small black dots located in the places, transitions appear as bars or rectangles, and arcs are simply arrows connected the system of places and transitions. A fifth element used to describe a Petri net is the marking of the system, denoted by m. This term is used to describe the state of the system. A simple example of a Petri net, along with its marking, is shown in Figure 1.

In classical Petri net theory, each of the four basic elements provides a specific interpretation of the system. Places may be used to represent locations within the system where a part (which can be represented by a token) may be represented as

a part at the location awaiting processing. Examples of places include machines, buffers, or inspection points. The tokens in the system hold the truth of the place in which they are located (i.e. part is at machine A, etc.) Transitions represent the actual process or event that a part is subjected to as it moves through the system [Moore, 1995]. A sequence of transitions can represent the events occurring in a dynamic system.

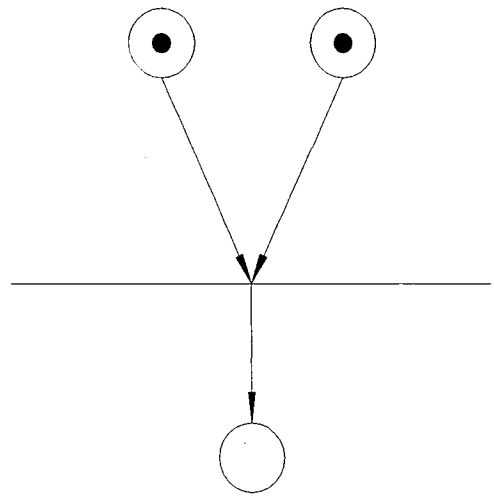


Figure 1 – Example of Basic Petri Net

The arcs are used to show the path that the objects are capable of taking through the system, the direction being given by the arrowhead of the arc. It is possible for each transition to have a number of input and output arcs in order to show the conditions both before and after the transition or the resources that were released or created during the event.

In order for the tokens to move throughout the system, the transitions must be "fired." Before a transition is fired, it must be enabled. For a transition to be enabled, each one of the input places to the transition must contain at least one token [Odrey, 2001]. If any one of the input places does not contain any tokens, than the associated transition will not be able to fire. When the transition fires, it takes a specific number of tokens from each of its input places, and passes tokens to each of its output places. The number of tokens that are passed is denoted by the weight of the arc connecting the places and transitions. Typically, only one token is passed and there is no number written above the arc. If multiple tokens are passed, however, the number of tokens will be written above the arc.

2.2 Types of Petri Nets

There are several different types of Petri nets that have been developed, the most basic of which is the pure Petri net. This is a net in its simplest form. In order to be classified as a pure Petri net, the net must contain no self-loops. To meet this requirement, no place in the net can be an input and an output to a transition simultaneously.

Another common set of Petri nets are called timed Petri nets. In a basic Petri net, there is no representation of the time that an actual operation takes within the system. One approach is to incorporate a delay function after a transition becomes enabled and before firing. Graphically, this delay is represented by a thick black bar,

rather than the thin black bar used in basic nets [Desrochers, 1995]. As opposed to timed transitions, time duration can also be signified for a place by identifying instantaneous time on transitions (instantaneous transitions) before and after the place.

A third set of Petri nets is termed stochastic Petri nets. These were developed to accommodate variable transition times [Molloy, 1982]. The transition times for the net can follow a number of different mathematical distributions, e.g. an exponential or Gaussian distribution. Whichever distribution is followed, it determines the firing times for all of the transitions in that particular net. A stochastic Petri net is commonly used to model events such as failure and repair times for individual machines [Desrochers, 1995]. Graphically, the transitions in stochastic Petri nets are represented by a thick white box.

A group of nets related to stochastic Petri nets are termed generalized stochastic Petri nets [Desrochers, 1995]. This group contains Petri nets that have both the immediate transitions of a Petri net as well as stochastic transitions. Nets of this type are used to represent systems that contain both types of processes (instantaneous and distributed.) The most common type of distribution in this group is the exponential distribution [Desrochers, 1995].

For the purpose of this thesis, we will use a variation of timed Petri nets called real-time Petri nets. The main reason for using a real-time Petri net, as opposed to a basic timed Petri net, is the ability to both emulate and control a system simultaneously [Peng, 2003]. The timing for this version of a net will be handled by a timing delay vector, which is discussed in Chapter Three.

2.3 Properties

When analyzing Petri nets, there are two types of properties that can be considered: behavioral and structural. Behavioral properties are those properties that are dependent on the initial marking of the system, while structural properties are those properties that are dependent on the structure of the system model. The following is a list of the most common behavioral and structural properties, along with their definitions [Desrochers, 1995]:

Behavioral:

1. Boundedness: the number of tokens present in a system remains the same after a firing sequence.
2. Liveness: the Petri net contains no deadlocks, or markings that will not allow any more transitions to fire. There are five different levels of liveness:
    a. Level 0: the transition can never be fired.

b. Level 1: the transition is potentially fireable.

c. Level 2: for every finite positive integer *n* there exists a firing sequence that contains a transition at least *n* times.

d. Level 3: there is an infinite-length firing sequence in which the transition occurs infinitely often.

e. Level 4: the transition is level 1 live for every marking in the reachability tree.

3. Reversibility: for every marking that can be obtained from the initial marking, there is a path that will return the system to its initial marking.
4. Persistence: for every two enabled transitions, the firing of one will not prohibit the firing of the other.
5. Reachability: a marking m_i is reachable from m_o by way of a series of existing transitions in the net.
6. Conservative: the weighted sum of the tokens in every node of the net is a constant value.
7. Strictly Conservative: the weighted sum of the tokens in every node of the net is constant for all reachable markings.
8. Consistency: the coverability tree has a circuit that contains each transition at least once.
9. Repetitive: the coverability tree has a circuit that contains all the transitions an infinite number of times.

Structural:

1. Structurally Live: the net is live for any finite initial marking.
2. Completely Controllable: any marking is reachable from initial marking m_o.

The properties that are to be used in the research presented here are boundedness, liveness, and reversibility. The reasons for choosing these properties are as follows:

1. Boundedness: this property will be evaluated to ensure that no tokens are created or destroyed in the firing of the transitions. This ensures that the net fires in a sequential manner.
2. Liveness: this property will be evaluated to ensure that there are no deadlocks that would prevent the system from continuing to fire (L-4 liveness).
3. Reversibility: the property will be evaluated to ensure that the CPU of the PLC scanning process is properly represented as a sequential process.

The other behavioral properties are not evaluated in this research because of the basic nature of the net. The structural properties are not evaluated because they are beyond the scope of the method proposed in this thesis.

Chapter 3 – Sensor-Based Stage Petri Nets

3.1 Introduction

In today's world, the most common method of executing automation tasks, particularly in the area of manufacturing, is the use of programmable logic controllers, or PLCs. Their widespread use is due to their toughness, relatively low cost, and ease of programming [Peng, 2003]. There are many common programming languages used in PLCs, the most common being instruction lists, ladder logic diagrams, function block diagrams, sequential function charts, and structured text [Lewis, 1998]. Of these control languages, the most commonly used is the ladder logic diagrams, or LLDs.

A ladder logic diagram is a means of graphically representing the logic that is required in the automated system [Chang, 1998]. The diagram itself consists of two rails of the ladder and various rungs that represent the logic being programmed. An example of a ladder logic diagram is shown in Figure 2.

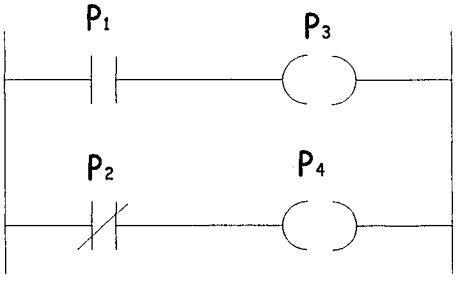


Figure 2 – Ladder Logic Diagram Example

This diagram is a simple example of ladder logic. The output is only a function of the input. The first rung uses a normally open contact, and the state of output P_3 is the same as input P_1. On the second rung, output P_4 has the opposite condition of input P_2. For example, pushing switch P2 turns off output P4. Salvador Rojas-Murillo [Rojas-Murillo, 2000] has previously developed the ladder-logic diagrams for the Lehigh University flexible manufacturing cell.

Using these ladder diagrams, it is possible to represent a control process both sequentially and graphically. While the simplicity of programming a PLC using ladder logic diagrams is one of its biggest attributes, it is also one of its biggest downfalls. A complex control system involves parallel tasks which interact at times, and ladder logic doesn't provide much in the way of design contexts to deal with this occurrence in the system. This is where Petri nets produce an advantage in modeling a complex system.

Petri nets have been applied extensively to discrete event control systems, in areas such as design, specification, verifications, and evaluation of a system's performance [Peng, 2003]. The main advantage of Petri nets, especially when compared to ladder logic diagrams, is the ability to check a given system for the desired properties. The first instance of a discrete-event control system using Petri nets occurred in 1980 [Chocron, 1980]. Baker and Song were the first to propose the use of Petri nets as a substitute for ladder logic diagrams [Baker, 1992].

Zhou and Twiss [Zhou, 1998] considered the concept of controlling a water treatment facility using PLCs. In their study, they looked at two different methods of control over the facility. The first method involved creating a ladder logic diagram in order to program the PLC, and then deriving the associated Petri net. The second method was to develop a Petri net in order to program the PLC. Based on their work, they concluded that Petri net methods of diagramming a system were better than ladder logic diagrams in terms of legibility of the system diagram and flexibility of the control system. The major problem with the Petri nets that resulted from the PLC programming was that the net was constructed from the analysis of a process-based ladder diagram (all of the inputs and outputs are based on an actual process being performed), as opposed to being directly derived from a sensor-based ladder program (the control decisions are based on sensor inputs and outputs, rather than a process being complete). Because of this, the resulting net didn't relate to neither the ladder logic program nor the PLC's control behavior.

This is one example where the concept of stage-based programming can be used to allow the user to break a complex system into stages, where each stage is programmed individually with no regard for how the stages will affect the overall program. This allows for construction of a sensor-based stage Petri net, which can easily be translated into a ladder logic diagram.

3.2 Stage Programming of Discrete-Event Control System

Any manufacturing process can be viewed as a series of discrete events, and typically multiple events can occur at the same time [Peng, 2003]. One event can occur that will cause a series of events that take place either simultaneously or asynchronously. If an inappropriate event occurs, it is possible that the system will deadlock, at which point it will not function at all, or the system will run into a conflict within itself. Stage programming can serve as a very useful tool in designing deadlock-free discrete-event control systems [Peng, 2003].

The basic concept of stage programming is relatively simple. A program is broken down into logical stages. At each of these stages, all of the output sensors are dependent only on the input sensors, and the input sensor can be easily identified within the system. The interlocking programming process of a complex system is simplified by the use of independent I/O combinational logic between the individual stages. This combination logic says that each output sensor at each stage of the system is affected by one or more input sensors. This relationship can be expressed in the following ways:

Output sensor = f\{input sensor (1), input sensor (2), …input sensor (n)\}

or

$$Y_i = f\{X(1), X(2), \ldots X(n)\}.$$

In the equation above, Y represents the binary status of an output sensor and X represents the binary status of an input sensor, while n is simply a positive integer index [Peng, 2003]. The status of each sensor is classified as binary because it can only have one of two values, either open or closed (on or off). For example, a part sensor located on the conveyor can either be open (no part present) or closed (part present). For an analog sensor, such as a force-torque sensor, the sensor output is converted to binary by setting a threshold value. For instance, for a force-torque sensor with a threshold value of 35 ft-lbs of torque, a value less than 35 ft-lbs would have a sensor output of open, while above this value it would be read as closed.

In actual PLC stage programming, the CPU scans each of the input stages sequentially [Peng, 2003], and then updates the output stages sequentially. According to this process, the output sensor at the ith stage is controlled by two parameters, S_i and X_i. S_i is the status of that particular stage place, and X_i is the combinational logic function of the input sensors for that stage. By combining the S_i and X_i terms, the following equation is formed for the output status of a given sensor:

$$Y_i = S_i \times f\{X(1), X(2), \ldots X(n)\} \tag{1}$$

where Y_i is the output status of a sensor, and has a value of either 0 or 1, and S_i is a stage place indicator to enable the input logic combination at that stage. Put another way, a value $S_i = 1$ enables the I/O logic at the ith stage, while a value of 0 disables

the logic. In using this concept, equation (1) can be used to define the control logic for each individual stage's I/O. By integrating all of these equations for all given stages in a system, it is possible to construct a complete PLC stage program. A complete example of this integration for a given system is presented in Chapter Four.

3.3 Sensor-Based Stage Petri Net

Use of a real-time Petri net (RTPN) tries to allow for both control and simulation of a particular system [Zhou, 1998]. A real-time Petri net controls a discrete-event control system by reading the input sensors in a sequential manner and using this information to update the output sensors, again in a sequential manner. These output sensors can then trigger the appropriate event to occur under the proper conditions. In order to form a real-time Petri net (RTPN) to perform the function of an untimed Petri net, a timed Petri net can be associated with the input and output sensor information. This combination yields the following eight-tuple [Venkatesh, 1994], which is an augmentation of a basic Petri net (P, T, I, O, m_0):

$$RTPN = (P, T, I, O, m_0, D, X, Y),$$

where

P: finite set of places;

T: finite set of transitions, with $P \cap T = 0$ and $P \cup T \neq 0$;

$P \times T \rightarrow N$: input function that specifies arcs directed from places to

transitions, with N being the set of natural numbers;

P x T → N: output function that specifies arcs directed from transitions places;

m_0: P → N: initial marking whose *i*th component represents the number of tokens, represented by dots in the *i*th place;

D: T → R^+: firing-time delay function, where R^+ is the set of non-negative real numbers;

X: P → B: input signal function, where B is the set of Boolean expressions of input addresses;

Y: T → {0, 1}: output signal function.

As stated earlier, a basic Petri net consists of five elements (P, T, I, O, m_0). The last three variables (D, X, Y) are added to the original five-tuple in order to be able show the time delay associated with the CPU scanning process and the status of the sensor inputs and outputs. These elements are vital to the formation of the sensor-based stage Petri net.

Notes on added variables:

(1) Vector D is a timing delay vector that models delays and synchronizations in the system and assigns delays to transitions by assigning a real number to the transitions.

(2) Vector X is used to enable a transition to fire. When the function associated with it is true and all input places have tokens, the firing rule can be executed. The associated functions and how they are developed will be discussed later.

(3) Vector Y is the vector that sends the output signals to the digital signal interface. When p_i has a token, the output function occurs. Place p_i can only write to a single element (y_i) of Y, therefore it can only be associated with a single output.

In the actual implementation of a sensor-based stage Petri net, the input functions are associated with places, and the output functions with transitions. X associates attributes to every place in the system, $X_i = X(p_i)$. The place p_i represents the input channel number associated with that particular place. Every input channel X_i can only have a value of 0 or 1. In a similar manner, $Y_i = Y(t_i)$ is the attribute associated with transition t_i, which represents the value that is to be sent to the output digital interface [Peng, 2003]. These values can also only be 0 or 1.

3.4 Stage Control Net

The addition of two more vectors (p_{Si} and t_{Si}) to the real-time Petri net allows for the formation of the sensor-based stage Petri net. These vectors form the stage control net (SCN) of the system. The vectors are:

(1) p_{Si}: stage priority place vector. This vector scans the I/O logic combination of the real-time Petri net at the *i*th stage. Once the logic is met, passing of this token activates a self-loop that tells the output place to begin firing.

(2) t_{Si}: transition signal vector. This vector moves a token through all of the places in the stage control net to monitor and update the signals to the I/O digital interface of the real-time Petri net controller.

The purpose of the stage control net is to monitor the status of the real-time Petri net. Because of this, only one token is allowed to pass through the stage control net. This can be written formally as:

$$O(p_{Si}, t_{Si}) = I(p_{Si}, t_{Si}) = 1.$$

The status of each I/O stage is checked by the t_{Si} vector of the stage control net once its token moves to that stage. If no token is present, then no action is enabled at that stage. In addition, even if there is a token present, the action will not be enabled if the input sensor values cannot validate the combinational logic for that stage.

By adding these two vectors to a real-time Petri net, we can formally define a sensor-based stage Petri net (SBSPN) in the following manner:

SBSPN = (SCN) U (RTPN)

where

{P, T, I, O, m_0, D, X, Y} is the real-time Petri net (RTPN) and where the stage control net is SCN = {p_{Si}, t_{Si}} as defined earlier.

3.5 Input Contacts

There are two types of input contacts that are used in the discrete-event control system design considered in this thesis for the flexible manufacturing system at Lehigh University: normally open and normally closed. These are denoted as NO and NC, respectively. A token present in either means that particular sensor has been activated. The different types of contacts are shown in Figure 3.

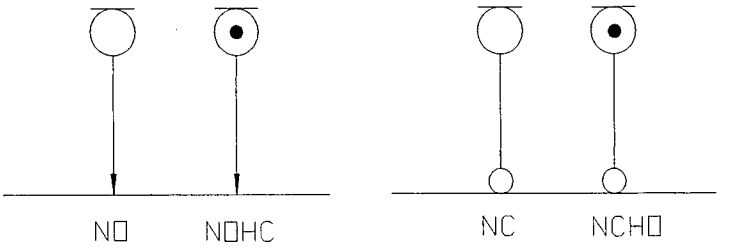


Figure 3 – Types of Contacts

The normally open/holding closed place (NOHC) with a directed arc represents a sensor that is normally open but holds in a closed position once it is

activated during an operation. For example, a token present in the NOHC place means that the contact is activated and will pass a signal to the control circuit. Similarly, the normally closed/holding open (NCHO) with an inhibitor arc means that the sensor is normally closed but holds to open once activated [Peng, 2003]. The use of the inhibitor arc (represented by a small circle touching a transition) prevents a transition from being fired when its input places have as many tokens as the weight of the inhibitor arc. As an example, a token in the NOHC allows the signal to pass through, while a token in the NCHO means that the signal will disconnect. The status of all the places in the system depends on the on/off status of these contacts.

3.6 Summary

The use of combinational logic, defined in section 3.2 as the output sensor of each stage being a function of the status of each stage (S_i) and the input sensors at the same stage, simplifies the design of each stage of the system. Once all of the individual stages have been designed, the integration of all the single stations allows for the formation of complete discrete-event control system in a much simpler manner than other methods [Peng, 2003]. The use of this method also allows for easier analysis and evaluation of the control system. The following chapter will show the use of this method for two scenarios in the control of the Lehigh University flexible manufacturing cell.

Chapter Four – Sensor-Based Stage Petri Net Modeling in the Lehigh University Flexible Manufacturing Cell

4.1 System and Scenarios

The Lehigh University flexible manufacturing cell consists of the following equipment: one PUMA robot with a force/torque sensor, one conveyor system, one Rhino robot, one CNC machine with pneumatic vise, various sensors and a computer vision system. A drawing of the layout (not to scale) is presented in Figure 4.

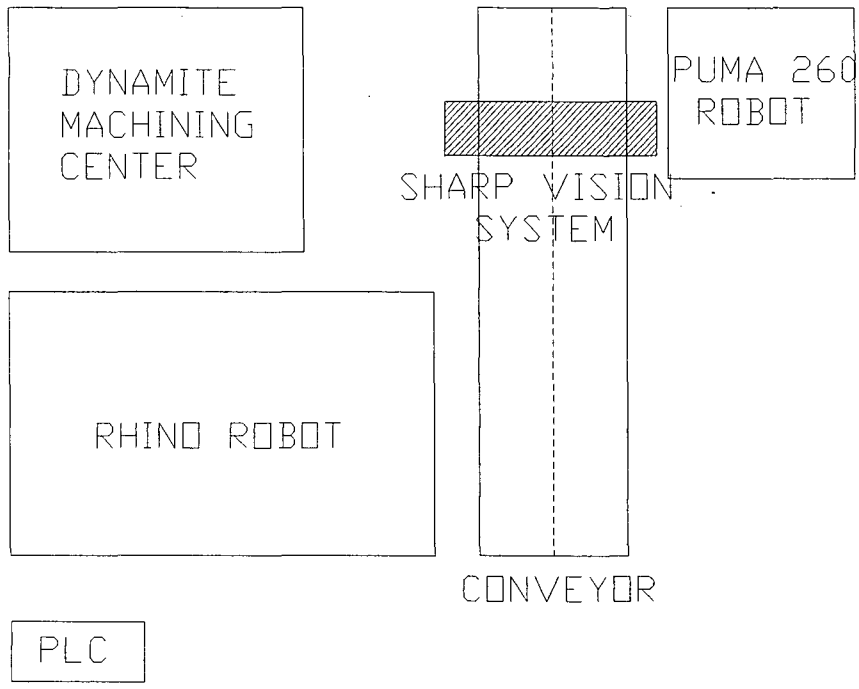


Figure 4 – FMS Layout (not to scale)

For the purpose of this paper, two different scenarios are considered, each scenario having its own set of operations to be performed:

Scenario One: The PUMA robot places a part on the conveyor. The conveyor then moves the part to the other side of the cell where the Rhino robot picks up the piece and places it in the vise of the CNC machine. The NC code is executed, and the part is machined. Once completed, the Rhino removes the part from the CNC machine and places it back on the conveyor. The conveyor then moves the part to the original side of the cell, where the PUMA removes the part and places it in storage.

Scenario Two: The second scenario is as follows: the PUMA places a part on the conveyor. The conveyor moves the part to the far side of the cell where the Rhino grips the part and places it in the vise of the CNC machine. A different NC code is executed, and afterwards the Rhino removes the part and places it back on the conveyor. As the conveyor is moving the part back to the load/unload side of the cell, the vision system scans the part. The PUMA then removes the part, places it in a fixture, and retrieves a bolt. It then places the bolt in the part, before moving the part to storage.

There are several assumptions that are made in regards to these two different scenarios:

1. The internal program for the PUMA to place a part on the conveyor is a separate program, and is not considered.
2. Each of the two parts has a distinctive shape when compared. This assumption is to ensure the vision system will choose the proper piece (either part A or part B) to have the bolts inserted by the PUMA.
3. Once the parts are removed from the system by the PUMA, they are placed in a box, on a pallet, or any other such device. Exactly what container is used is not relevant to the rest of the cell.

4.2 Inputs and Outputs

In order to implement a sensor-based stage Petri net model for this system, the proper sensors must be incorporated into the system. Table 1 lists the input and output sensors, the sensor places, and a description of each one included in the cell. Figure 5a shows the manufacturing cell and the relative locations of each of the input sensors, while Figure 5b shows the cell with locations of the output sensors.

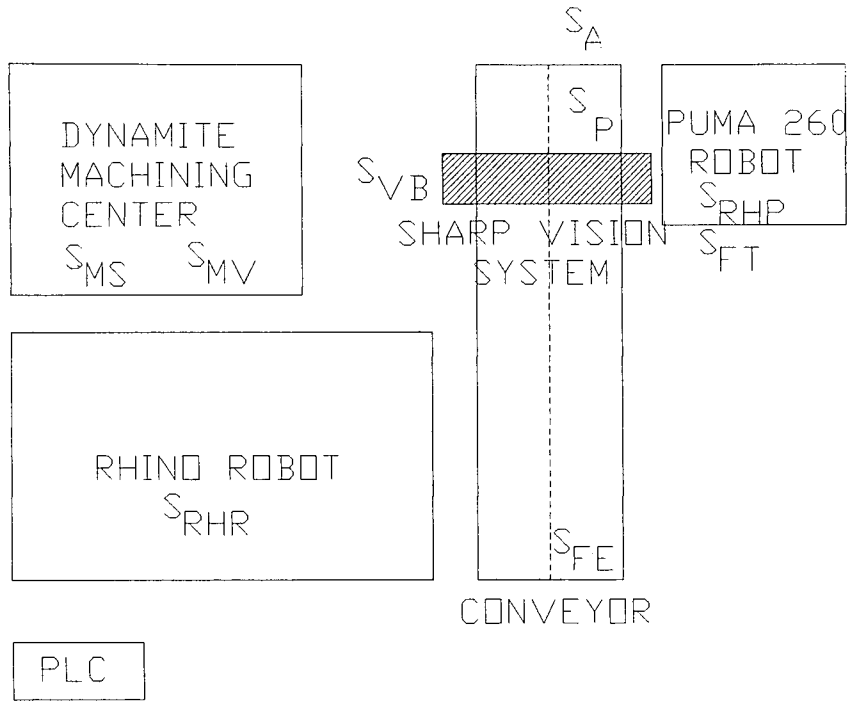


Figure 5a – Cell Layout with Input Sensor Positions

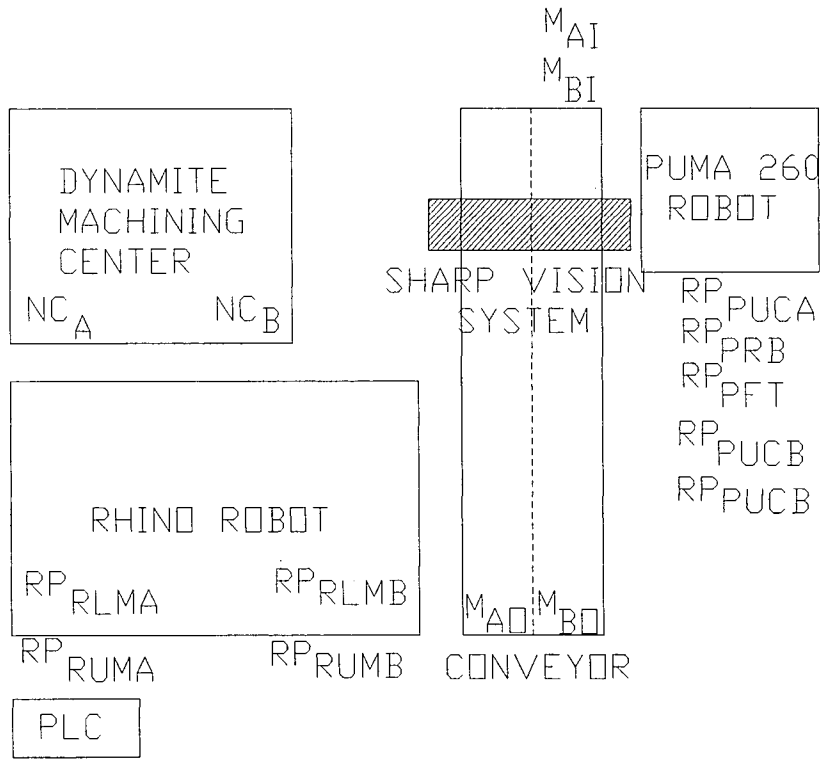


Figure 5b – Cell Layout with Output Sensor Positions

Inputs	Sensor Places	Description
S_p	P_1	Conveyor part sensor
S_{RHP}	P_2	PUMA robot at-home sensor
S_{FE}	P_3	Conveyor at far end of cell
S_{MS}	P_4	CNC machine spindle sensor
S_{MV}	P_5	CNC machine vise sensor
S_{RHR}	P_6	Rhino robot at-home sensor
S_A	P_7	Conveyor at starting position
S_{VB}	P_8	Vision system camera
S_{FT}	P_9	Force/torque sensor on PUMA

Outputs	Sensor Places	Description
M_{AI}	P_{A1}	Part A input motor
RP_{RLMA}	P_{A2}	Rhino loads CNC center with part A
NC_A	P_{A3}	CNC center NC code to machine part A
RP_{RUMA}	P_{A4}	Rhino unloads part A to conveyor
M_{AO}	P_{A5}	Part A output motor
RP_{PUCA}	P_{A6}	PUMA unloads part A from conveyor
M_{BI}	P_{B1}	Part B input motor
RP_{RLMB}	P_{B2}	Rhino loads CNC center with part B
NC_B	P_{B3}	CNC center NC code to machine part B
RP_{RUMB}	P_{B4}	Rhino unloads part B to conveyor
M_{BO}	P_{B5}	Part B output motor
RP_{PUCB}	P_{B6}	PUMA unloads part B from conveyor
RP_{PRB}	P_{B7}	PUMA retrieves bolt
RP_{PFT}	P_{B8}	PUMA inserts bolt
RP_{PUCB}	P_{B9}	PUMA removes part B from fixture
CPU Scans	P_{Si}	Stage Control Net *i*th stage place, i=1-15

Table 1 – Inputs & Outputs for the System

4.3 Logic Combinations and Boolean Equations

The first step in creating a sensor-based stage Petri net model is to develop the control logic at each individual stage of the system. This is done by using the I/O sensors at each stage that are relevant for the given process to construct the logic combinations for each stage.

The first process to consider is scenario 1, as described earlier. Using the sensor inputs and outputs, the logic combinations and associated verbal descriptions of the equations for this process are (note: these equations involve regular multiplication):

$$M_{AI} = S_p \times S_{RHP} \tag{2}$$

(the part A input motor (M_{AI}) is activated when the conveyor part sensor (S_P) and the PUMA at-home sensor (SRHP) are activated)

$$RP_{RLMA} = S_{FE} \times S_{MS} \times S_{MV} \times S_{RHR} \tag{3}$$

(the Rhino loads the CNC machining center (RP_{RLMA}) when the conveyor is at the far end of the cell (S_{FE}), the CNC spindle isn't moving (S_{MS}), the CNC vise is open (S_{MV}), and the Rhino is in the home position (S_{RHR}))

$$NC_A = S_{RHR} \times S_{MS} \times S_{MV} \quad (4)$$

(the NC code for part A is performed (NC_A) when the Rhino is in the home position (S_{RHR}), the CNC spindle isn't moving (S_{MS}), and the vise is closed S_{MV}))

$$RP_{RUMA} = S_{RHR} \times S_{MS} \times S_{MV} \times S_{FE} \quad (5)$$

(the Rhino unloads part A from the CNC machine (RP_{RUMA}) when the Rhino is in the home position (S_{RHR}), the spindle is stopped (S_{MS}), the vise is open (S_{MV}), and the conveyor is at the far end of the cell (S_{FE}))

$$M_{AO} = S_P \times S_{RHR} \quad (6)$$

(the part A output motor is activated (M_{AO}) when the conveyor part sensor is activated (S_p) and the Rhino is in the home position (S_{RHR}))

$$RP_{PUCA} = S_A \times S_{RHP} \quad (7)$$

(the PUMA unloads part A from the conveyor (RP_{PUCA}) when conveyor is in the starting position (S_A) and the PUMA is in the at-home position (S_{RHP}))

Once the logic combinations have been developed, these are then converted into real-time Petri net Boolean equations by substituting the sensor places for the appropriate inputs and outputs according to Table 1. Using this method, the logic combinations listed above become:

$$P_{A1} = P_1 \times P_2 \quad (2)$$

$$P_{A2} = P_3 \times P_4 \times P_5 \times P_6 \quad (3)$$

$$P_{A3} = P_6 \times P_4 \times P_5 \quad (4)$$

$$P_{A4} = P_6 \times P_4 \times P_5 \times P_3 \quad (5)$$

$$P_{A5} = P_1 \times P_6 \quad (6)$$

$$P_{A6} = P_7 \times P_2 \quad (7)$$

Given that all logic combinations have been converted to their Boolean equivalents, a real-time Petri net can be constructed for each stage, along with the stage control net. An example of the real-time Petri net and stage control net for equation (2) is shown below in Figure 6. The RTPN and SCN for the entire process of manufacturing part A is shown in Appendix 1.

This diagram shows how both P_1 and P_2 must contain a token in order for P_{AI} to fire. Once this fires, a token is passed through the inhibitor arc to P_{ST1} of the stage control net.

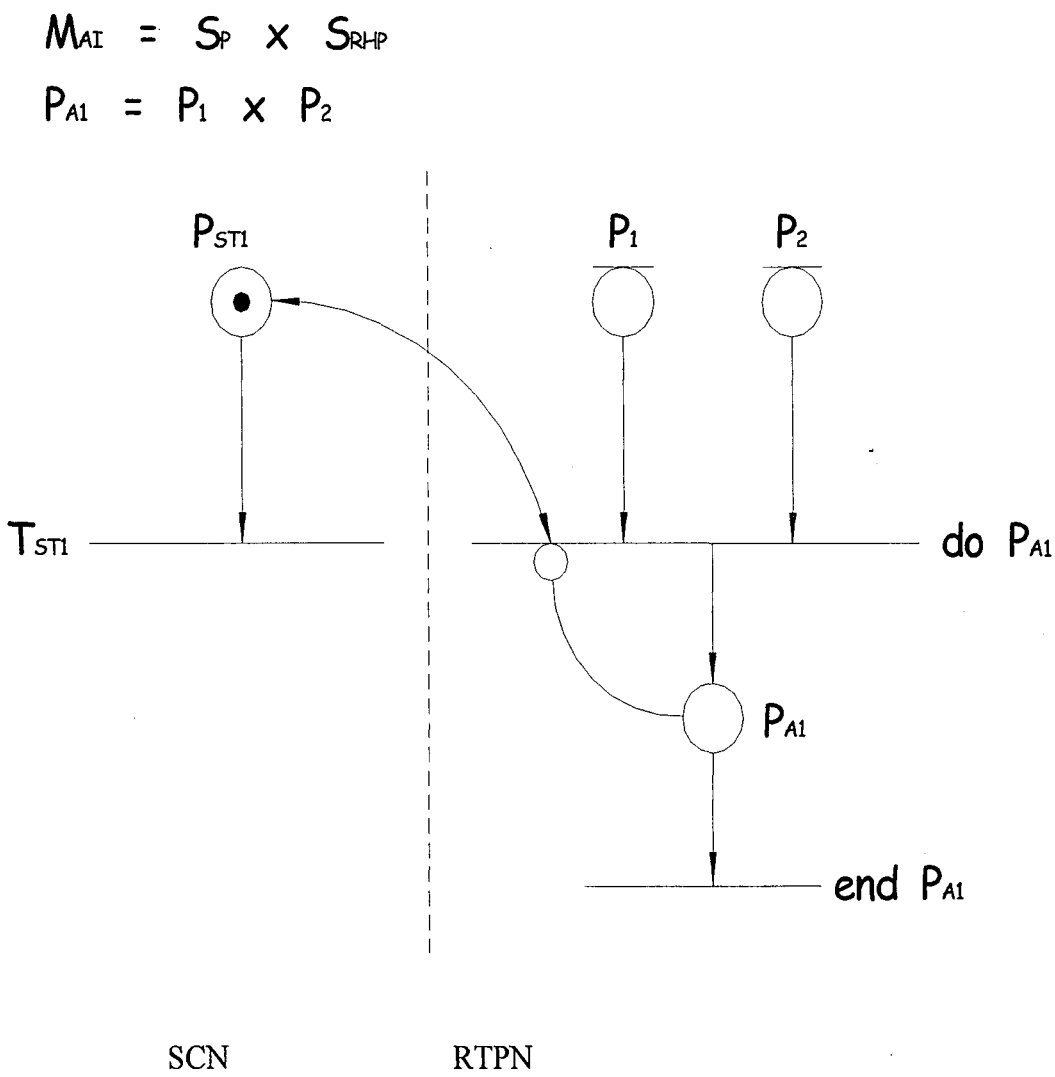


Figure 6 – SCN U RTPN for Equation (2)

Accordingly, scenario 2 contains the following logic combinations:

$M_{BI} = S_p \times S_{RHP}$ (8)

(the part B input motor (M_{BI}) is activated when the conveyor part sensor (S_P) and the PUMA at-home sensor (SRHP) are activated)

$RP_{RLMB} = S_{FE} \times S_{MS} \times S_{MV} \times S_{RHR}$ (9)

(the Rhino loads the CNC machining center (RP_{RLMB}) when the conveyor is at the far end of the cell (S_{FE}), the CNC spindle isn't moving (S_{MS}), the CNC vise is open (S_{MV}), and the Rhino is in the home position (S_{RHR}))

$NC_B = S_{RHR} \times S_{MS} \times S_{MV}$ (10)

(the NC code for part B is performed (NC_B) when the Rhino is in the home position (S_{RHR}), the CNC spindle isn't moving (S_{MS}), and the vise is closed S_{MV}))

$RP_{RUMB} = S_{RHR} \times S_{MS} \times S_{MV} \times S_{FE}$ (11)

(the Rhino unloads part B from the CNC machine (RP_{RUMB}) when the Rhino is in the home position (S_{RHR}), the spindle is stopped (S_{MS}), the vise is open (S_{MV}), and the conveyor is at the far end of the cell (S_{FE}))

$M_{BO} = S_P \times S_{RHR}$ (12)

(the part A output motor is activated (M_{BO}) when the conveyor part sensor is activated (S_p) and the Rhino is in the home position (S_{RHR}))

$RP_{PUCB} = S_A \times S_{RHP} \times S_{VB}$ (13)

(the PUMA unloads part B from the conveyor (RP_{PUCB}) when conveyor is in the starting position (S_A), the PUMA is in

the at-home position ($S_{RHP,}$ and the vision system recognizes part B S_{VB}))

$$RP_{PRB} = S_{RHP} \quad (14)$$

(the PUMA retrieves a bolt (RP_{PRB}) when the PUMA is in the at home position (S_{RHP}))

$$RP_{PFT} = S_{FT} \quad (15)$$

(the PUMA inserts the bolt into part B (RP_{PFT}) when the force-torque sensor has not been activated (S_{FT}))

$$RP_{PUCB} = S_{FT} \quad (16)$$

(the PUMA removes part B and places it in storage (RP_{PUCB}) when the force-torque sensor has been activated (S_{FT}))

and the corresponding Boolean equations:

$$P_{B1} = P_1 \times P_2 \quad (8)$$

$$P_{B2} = P_3 \times P_4 \times P_5 \times P_6 \quad (9)$$

$$P_{B3} = P_6 \times P_4 \times P_5 \quad (10)$$

$$P_{B4} = P_6 \times P_4 \times P_5 \times P_3 \quad (11)$$

$$P_{B5} = P_1 \times P_6 \quad (12)$$

$$P_{B6} = P_1 \times P_2 \quad (13)$$

$$P_{B7} = P_2 \quad (14)$$

$$P_{B8} = P_9 \quad (15)$$

$$P_{B9} = P_9 \tag{16}$$

As for scenario 1, these Boolean equations are then transformed into the appropriate real-time Petri net and stage control net for each stage. All of these real-time Petri nets, similar to that shown in Figure 6, are included in Appendix B.

4.4 Integration of Stages

The final step in constructing a sensor-based stage Petri net model for the entire flexible manufacturing cell is the integration of all the real-time Petri nets and stage control nets. In order to integrate all of these and form the overall system representation, all of the real-time Petri nets and stage control nets are grouped into fifteen stages in the format of (stage control net) U (real-time Petri net). The fifteen stages are represented by the following unions:

$$(p_{S1}, t_{S1}) \cup (p_{A1} = p_1 \times p_2, t_{A1}) \tag{17}$$

$$(p_{S2}, t_{S2}) \cup (p_{A2} = p_3 \times p_4 \times p_5 \times p_6, t_{A2}) \tag{18}$$

$$(p_{S3}, t_{S3}) \cup (p_{A3} = p_6 \times p_4 \times p_5, t_{A3}) \tag{19}$$

$$(p_{S4}, t_{S4}) \cup (p_{A4} = p_6 \times p_4 \times p_5 \times p_3, t_{A4}) \tag{20}$$

$$(p_{S5}, t_{S5}) \cup (p_{A5} = p_1 \times p_6, t_{A5}) \tag{21}$$

$$(p_{S6}, t_{S6}) \cup (p_{A6} = p_7 \times p_2, t_{A6}) \tag{22}$$

$$(p_{S7}, t_{S7}) \cup (p_{B1} = p_1 \times p_2, t_{B1}) \tag{23}$$

$$(p_{S8}, t_{S8}) \cup (p_{B2} = p_3 \times p_4 \times p_5 \times p_6, t_{B2}) \tag{24}$$

$$(p_{S9}, t_{S9}) \cup (p_{B3} = p_6 \times p_4 \times p_5, t_{B3}) \tag{25}$$

$$(p_{SI0}, t_{SI0}) \cup (p_{B4} = p_6 \times p_4 \times p_5 \times p_3, t_{B4}) \quad (26)$$

$$(p_{SI1}, t_{SI1}) \cup (p_{B5} = p_1 \times p_6, t_{B5}) \quad (27)$$

$$(p_{SI2}, t_{SI2}) \cup (p_{B6} = p_1 \times p_2, t_{B6}) \quad (28)$$

$$(p_{SI3}, t_{SI3}) \cup (p_{B7} = p_2, t_{B7}) \quad (29)$$

$$(p_{SI4}, t_{SI4}) \cup (p_{B8} = p_9, t_{B8}) \quad (30)$$

$$(p_{SI5}, t_{SI5}) \cup (p_{B9} = p_9, t_{B9}) \quad (31)$$

The overall integration of the stage control nets mimics the CPU scanning process during system operation. This is the fact that allows the above unions to represent the entire system being modeled. When these equations are turned into a visual form, they form the complete sensor-based stage Petri net of the system. This overall net is shown in Figure 7. The input sensors are located down the middle of the net, while the process for part A is on the left and part B on the right. The stage control net can be seen encircling all of the processes of the system. In this form, the control net shows how it moves sequentially through all of the processes, reading the input sensors and updating the appropriate outputs.

Expansion of the system, such as adding more machines or other parts, is simply a matter of forming the correct logic combinations, converting them to Boolean equations, developing the stage control net and real-time Petri net for that component, and integrating it into the overall system. Contraction of the system is a simple case of removing the particular component from the net.

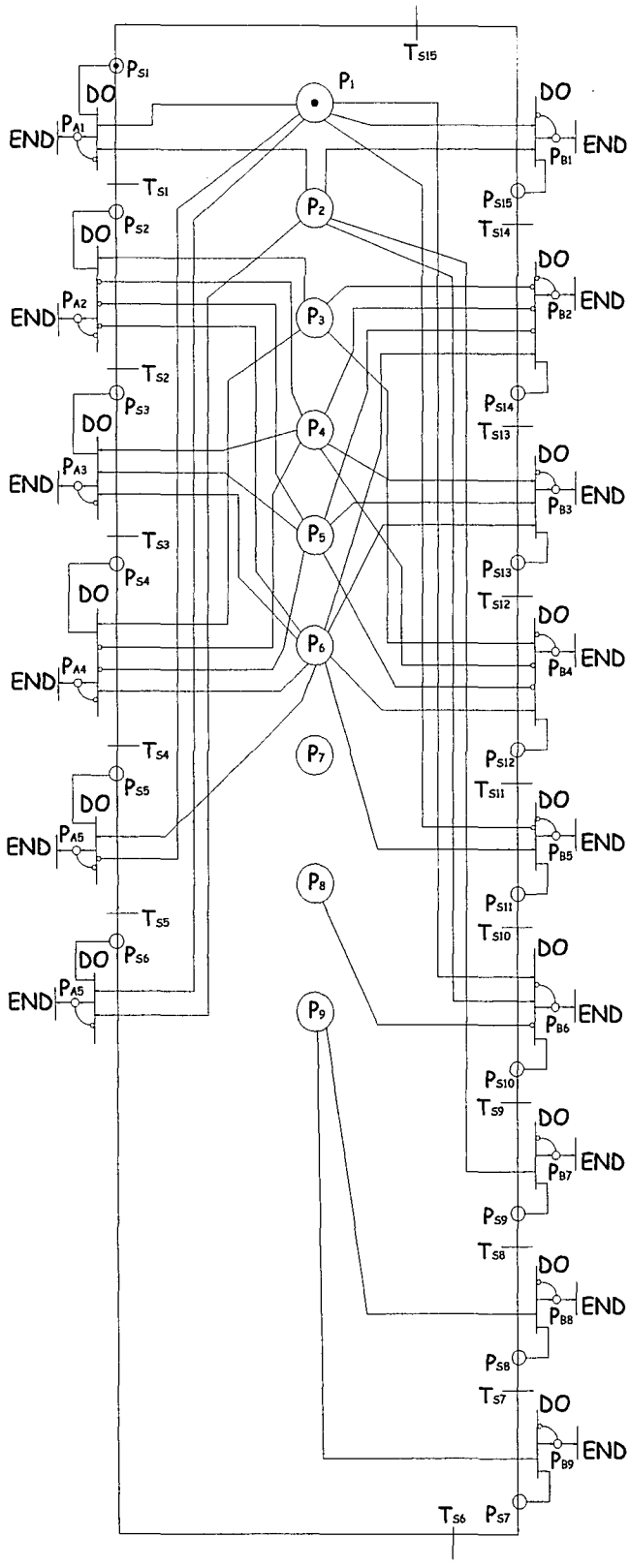


Figure 7 – SBSPN for Entire System

Chapter Five – Conclusions

Ladder logic diagrams were one of the first ways to program a PLC for an automation system. As time progressed, however, the systems being modeled became more and more complex, and the use of Petri nets helped to ease the PLC programming process.

By using a Petri net that uses sensor inputs and outputs, rather than the traditional manner of representing actual processes and resources, it becomes easier to program a complex system. A complex system can be broken into a series of logical stages. By analyzing the process of each stage, a series of equations representing the input and output sensor combinational logic for a particular stage can be determined. This combinational logic can then be converted into a corresponding set of Boolean equations, and these resulting equations can then be programmed into a PLC without worrying how one stage will react with another. The result of this process is an interlock-free control of the system.

Chapter Six – Future Work

The concept of integrating sensory input and output information with a timed Petri net results in a simplified manner of controlling and modeling a discrete-event system. This method can also be used to either add or subtract individual stages from the entire system without having to redo the entire net.

Because of the modularity of this method, possible future work would be to incorporate several more sensors into the system along with several more automated machines (robots, CNC machines, etc.) By incorporating more of this equipment, a greater range of parts would be able to be produced. Another way to increase the number of different parts would be to add other NC machine codes to the machining center.

References

1. Baker, H.A., Song, J., "A Graphical Simulation Tool for Programming Logic Controllers", Discrete Event Dynamic Systems – A New Generation of Modeling, IEEE Colloquium on Simulation & Control Applications, 1992, pp. 4/1-4/4.

2. Castillo, Ignacio, Smith, Jeffrey S., "Formal Modeling Methodologies for Control of Manufacturing Cells: Survey and Comparison". Journal of Manufacturing Systems, 21(1), 2002, pp. 40-57.

3. Chang, T., Wysk, R., Wang, H. Computer-Aided Manufacturing, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, NJ., 1998.

4. Chocron, D., Cerny, E., "A Petri Net Based Industrial Sequencer", Proceedings of the IEEE International Conference and Exhibition on Industrial Control and Instrumentation, 1980, pp. 18-22.

5. Desrochers, Alan A., Al-Jaar, Robert Y., Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis, IEEE Books, 1995.

6. Lewis, R.W., Programming Industrial Systems Using IEC 1131-1, Revised Edition, IIE Books, 1998.

7. Molloy, M. K. "Performance Analysis Using Stochastic Petri Nets", Proceeding of the IEEE. 77(4), 1982, pp. 541-580.

8. Moore, Kendra E., Brennan, John E., "Petri Nets and Simulation: A Tutorial", Proceedings of the 1995 Summer Computer Simulation Conference. July 24-26. Ottawa, Ontario, Canada. 1995

9. Odrey, Nicholas G., Green, Jonathan D., Appello, Adrienne., "A Generalized Petri Net Modeling Approach for the Control of Re-Entrant Flow Semiconductor Wafer Fabrication", Robotics and Computer Integrated Manufacturing 17, 2001, pp. 5-11.

10. Peng, Shihsen, Zhou, Mengchu, "Sensor-based Stage Petri Net Modelling of PLC Logic Programs for Discrete-Event Control Design", International Journal of Production Research. 41(3) 2003, 629-644.

11. Rojas-Murillo, Salvador. "Control of the Lehigh University Automated Manufacturing Cell Using Petri Nets", Master of Science in Industrial Engineering, Lehigh University. 2000.

12. Venkatesh, K., Zhou, Mengchu, Caudill, R. J., "Comparing Ladder Diagrams and Petri Nets for Sequential Controller Design Through a Discrete Manufacturing System", IEEE Transactions on Industrial Electronics, 41(6), 611-619

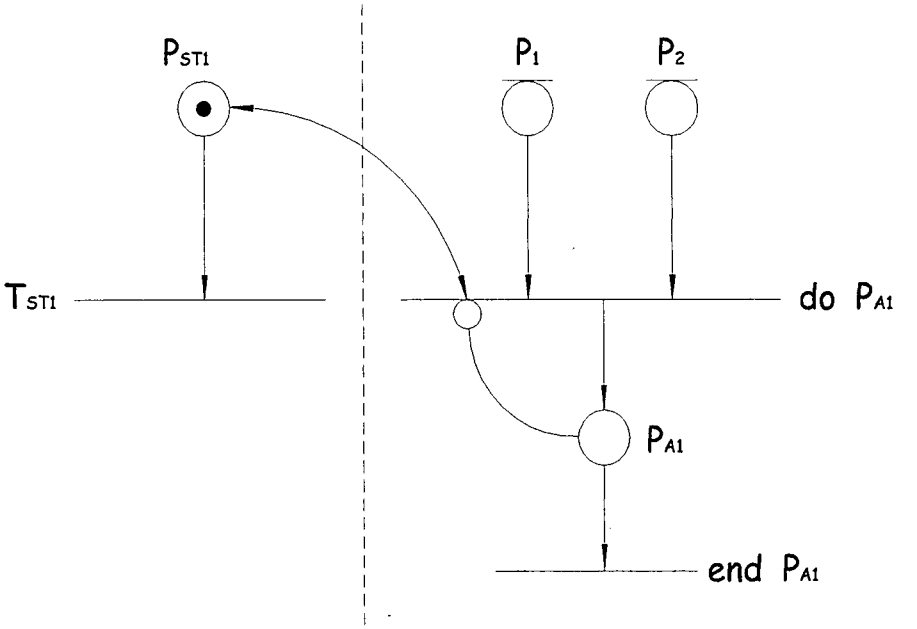
13. Zhou, Mengchu, Twiss, E., "Design of Industrial Automated Systems Via Relay Logic Programming and Petri Nets", IEEE Transactions on Systems, Man, and Cybernetics. 28(1), 1998, pp. 137-150.

14. Zhou, Mengchu, Venkatesh, K., Modelling, Simulation, and Control of Flexible Manufacturing Systems – A Petri Net Approach, Singapore – World Scientific, 1998.

Appendix A: Part A SCN and RTPN

(1) $M_{AI} = S_P \times S_{RHP}$

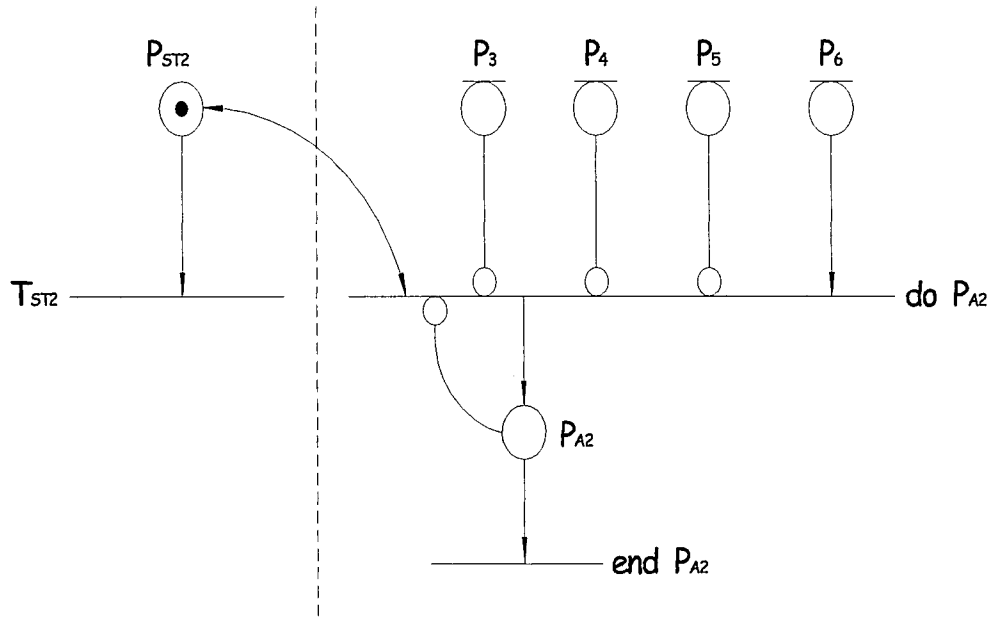
$P_{A1} = P_1 \times P_2$



Description: the part A input motor (M_{AI}) is activated when the conveyor part sensor (S_P) and the PUMA at-home sensor (SRHP) are activated.

(2) $RP_{RLMA} = \overline{S_{LA}} \times \overline{S_{V1}} \times \overline{S_{V2}} \times S_{RHR}$

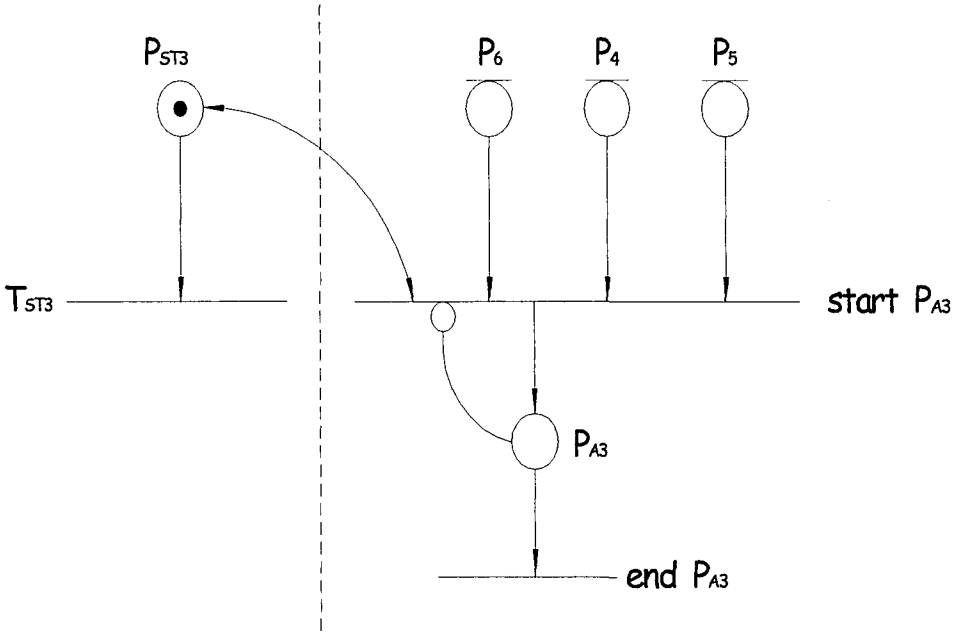
$P_{A2} = P_3 \times P_4 \times P_5 \times P_6$



Description: the Rhino loads the CNC machining center (RP_{RLMA}) when the conveyor is at the far end of the cell (S_{FE}), the CNC spindle isn't moving (S_{MS}), the CNC vise is open (S_{MV}), and the Rhino is in the home position (S_{RHR}).

(3) $NC_A = S_{RHR} \times S_{V1} \times S_{V2}$

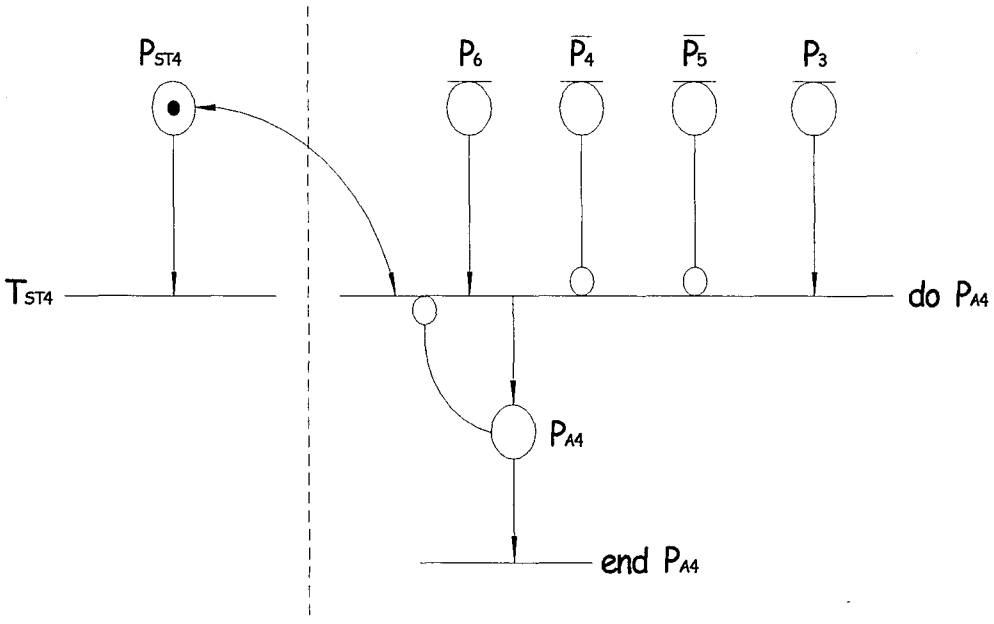
$P_{A3} = P_6 \times P_4 \times P_5$



Description: the NC code for part A is performed (NC_A) when the Rhino is in the home position (S_{RHR}), the CNC spindle isn't moving (S_{MS}), and the vise is closed S_{MV}).

④ $RP_{RUMA} = S_{RHR} \times \overline{S_{V1}} \times \overline{S_{V2}} \times S_{LA}$

$P_{A4} = P_6 \times P_4 \times P_5 \times P_3$

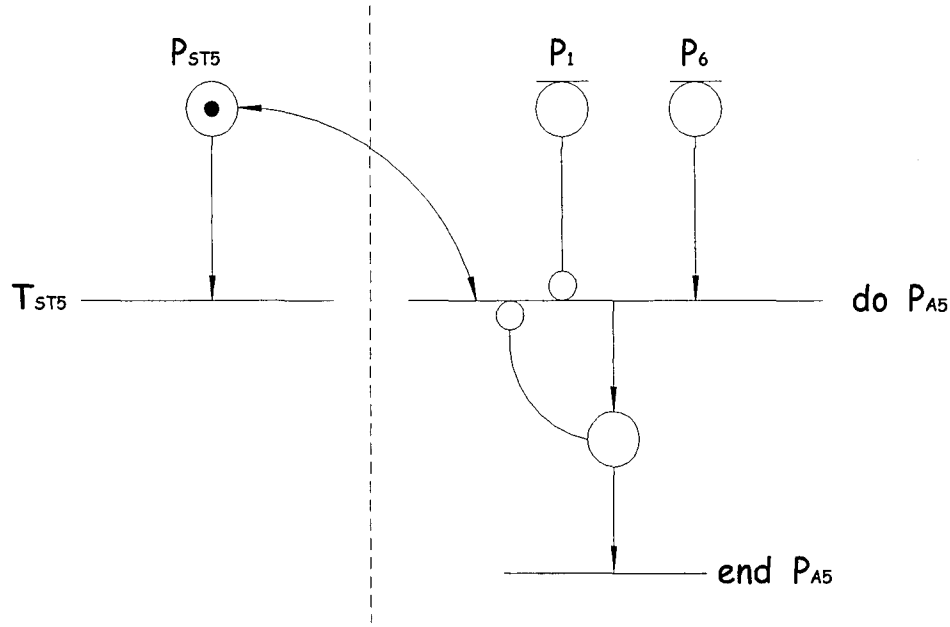


Description: the Rhino unloads part A from the CNC machine (RP_{RUMA}) when the Rhino is in the home position (S_{RHR}), the spindle is stopped (S_{MS}), the vise is open (S_{MV}), and the conveyor is at the far end of the cell (S_{FE}).

5

$M_{AO} = \overline{S_P} \times S_{RHR}$

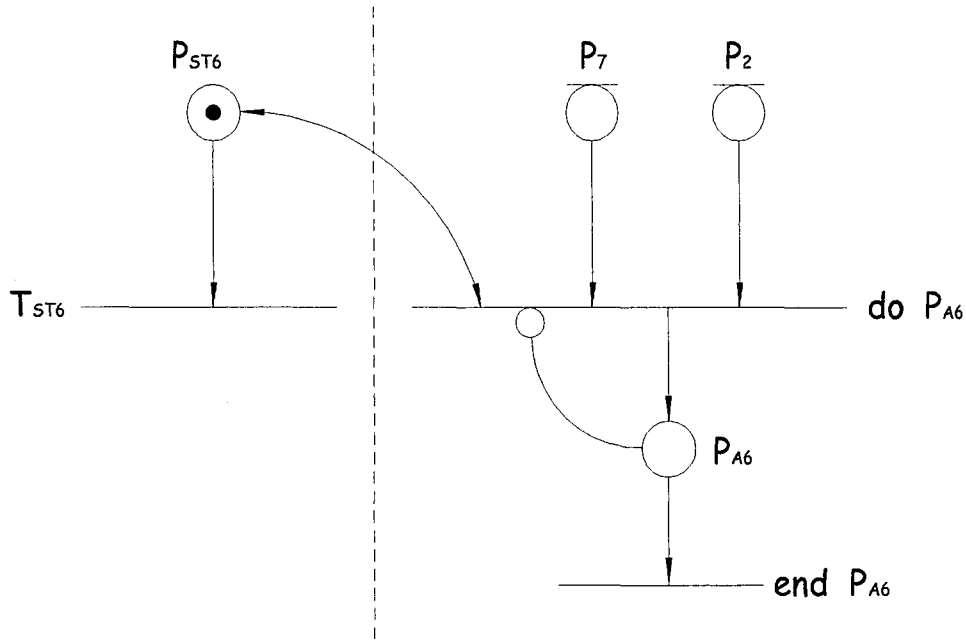
$P_{A5} = P_1 \times P_6$



Description: the part A output motor is activated (M_{AO}) when the conveyor part sensor is activated (S_p) and the Rhino is in the home position (S_{RHR}).

6 $RP_{PUCA} = S_A \times S_{RHP}$

$P_{A6} = P_7 \times P_2$



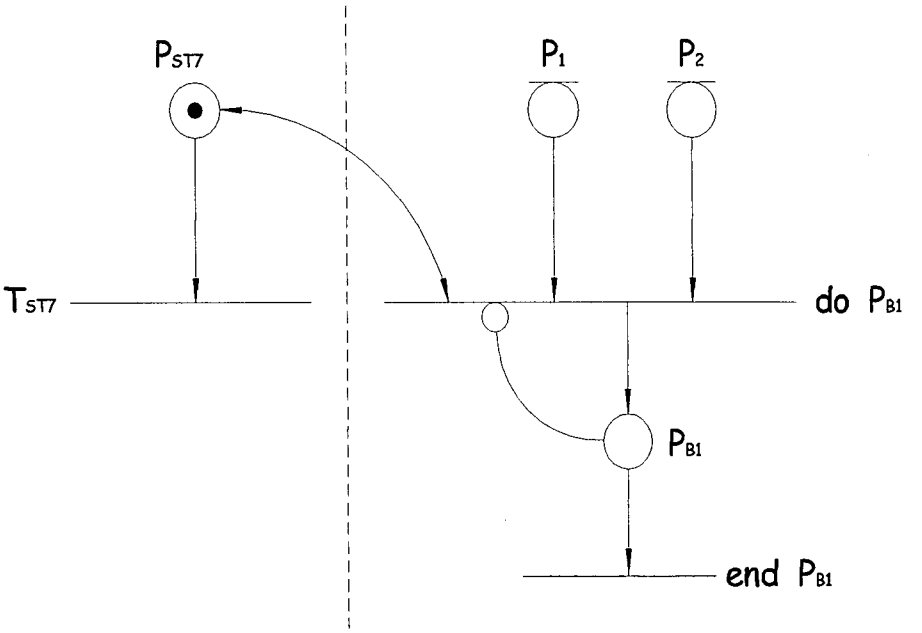
Description: the PUMA unloads part A from the conveyor (RP_{PUCA}) when the conveyor is in the starting position (S_A) and the PUMA is in the at-home position (S_{RHP}).

Appendix B: Part B SCN and RTP

7

$M_{B1} = S_P \times S_{RHP}$

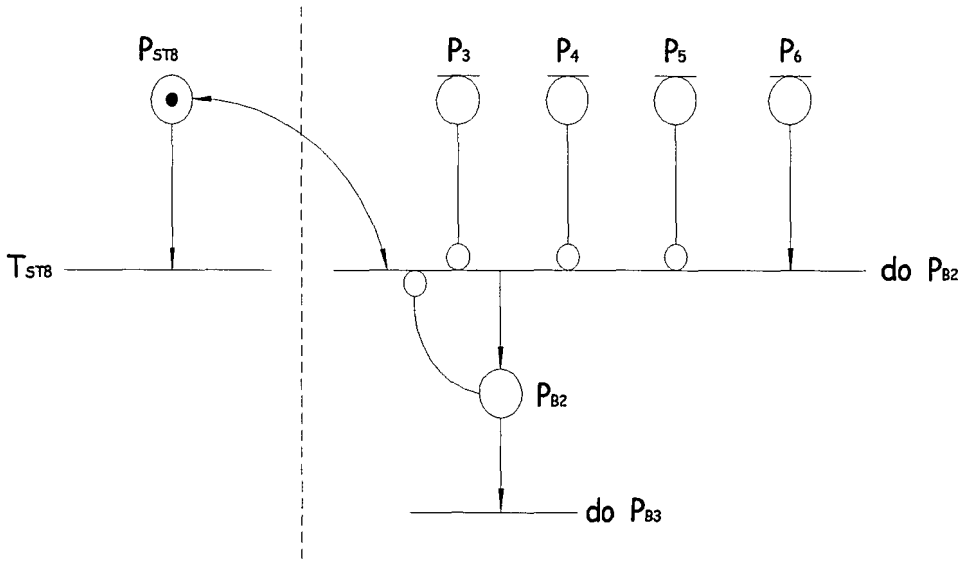
$P_{B1} = P_1 \times P_2$



Description: part B input motor (M_{BI}) is activated when the conveyor part sensor (S_P) and the PUMA at-home sensor (SRHP) are activated)

(8) $RP_{RLMB} = \overline{S_{LA}} \times \overline{S_{V1}} \times \overline{S_{V2}} \times S_{RHR}$

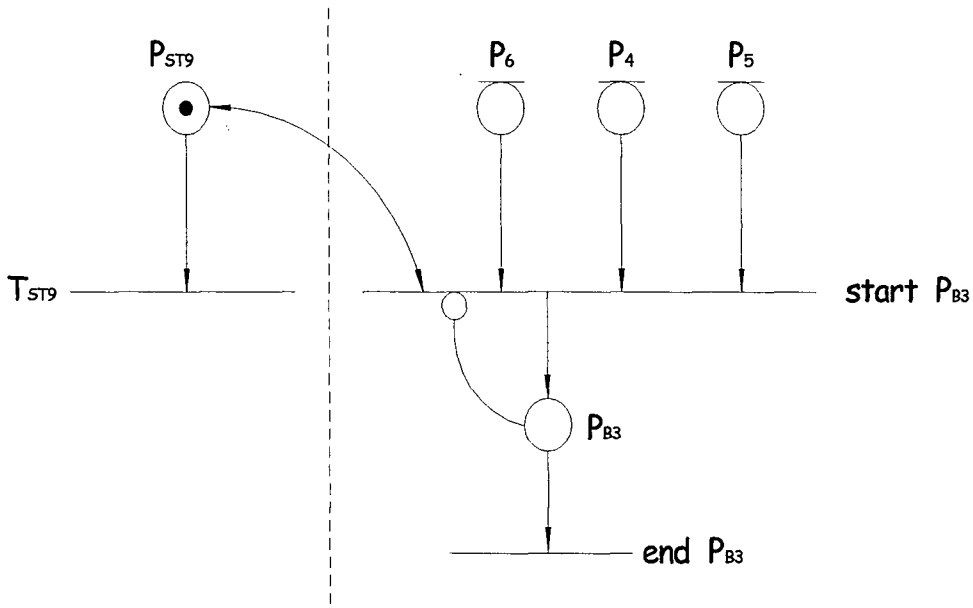
$P_{B2} = P_3 \times P_4 \times P_5 \times P_6$



Description: the Rhino loads the CNC machining center (RP_{RLMB}) when the conveyor is at the far end of the cell (S_{FE}), the CNC spindle isn't moving (S_{MS}), the CNC vise is open (S_{MV}), and the Rhino is in the home position (S_{RHR}).

9 $NC_B = S_{RHR} \times S_{V1} \times S_{V2}$

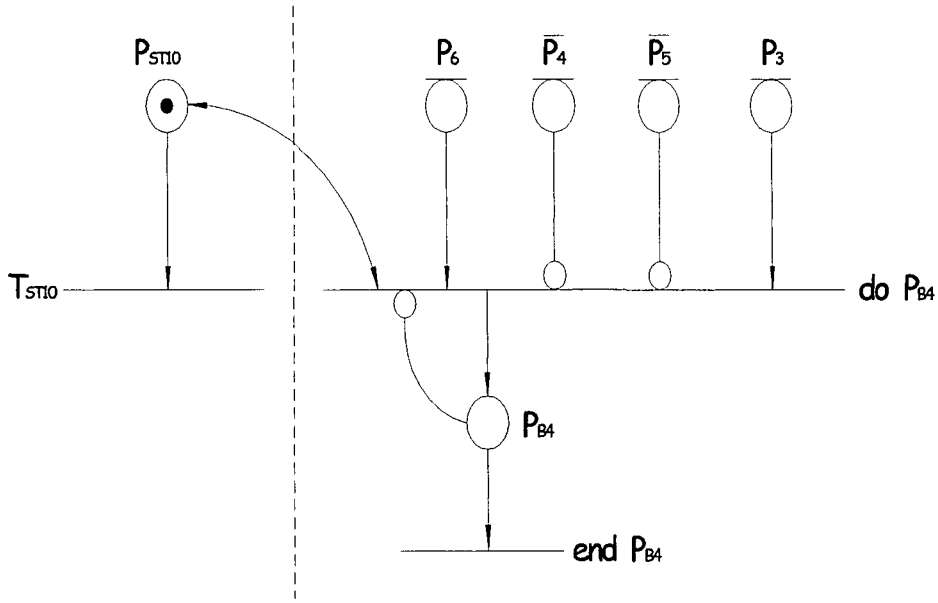
$P_{B3} = P_6 \times P_4 \times P_5$



Description: the NC code for part B is performed (NC_B) when the Rhino is in the home position (S_{RHR}), the CNC spindle isn't moving (S_{MS}), and the vise is closed S_{MV}).

(10) $RP_{RUMB} = S_{RHR} \times \overline{S_{M1}} \times \overline{S_{M2}} \times S_{LA}$

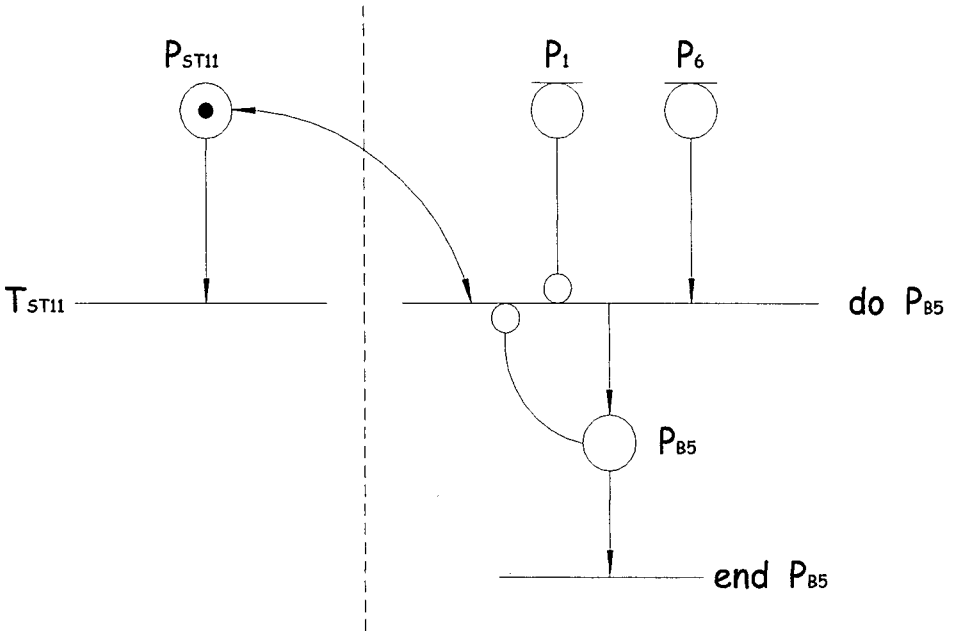
$P_{B4} = P_6 \times P_4 \times P_5 \times P_3$



Description: the Rhino unloads part B from the CNC machine (RP_{RUMB}) when the Rhino is in the home position (S_{RHR}), the spindle is stopped (S_{MS}), the vise is open (S_{MV}), and the conveyor is at the far end of the cell (S_{FE}).

11 $M_{BO} = \overline{S_P} \times S_{RHR}$

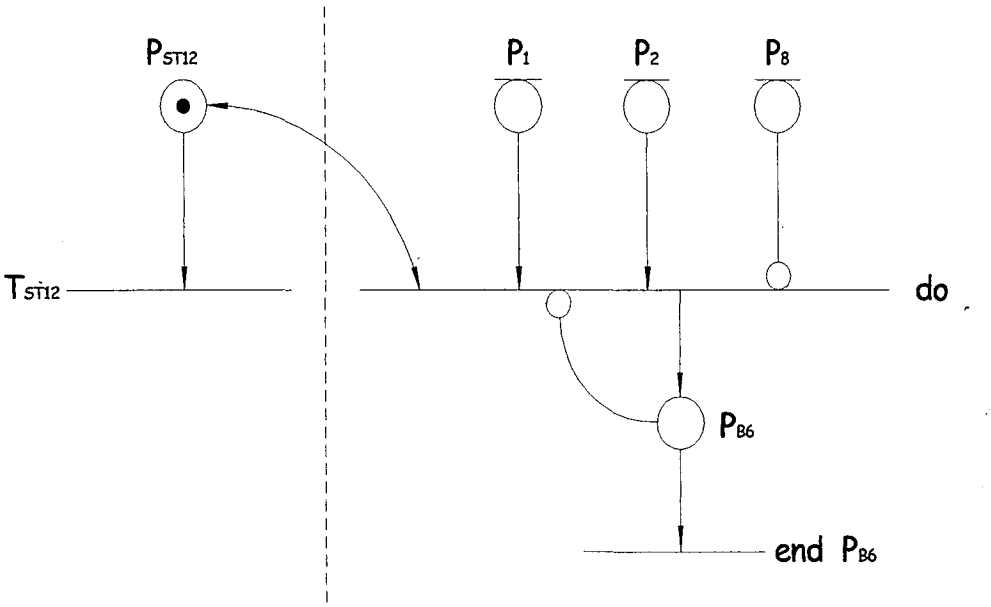
$P_{B5} = P_1 \times P_6$



Description: the part A output motor is activated (M_{BO}) when the conveyor part sensor is activated (S_p) and the Rhino is in the home position (S_{RHR}).

12 $RP_{PUCB} = S_A \times S_{RHP} \times \overline{S_{VB}}$

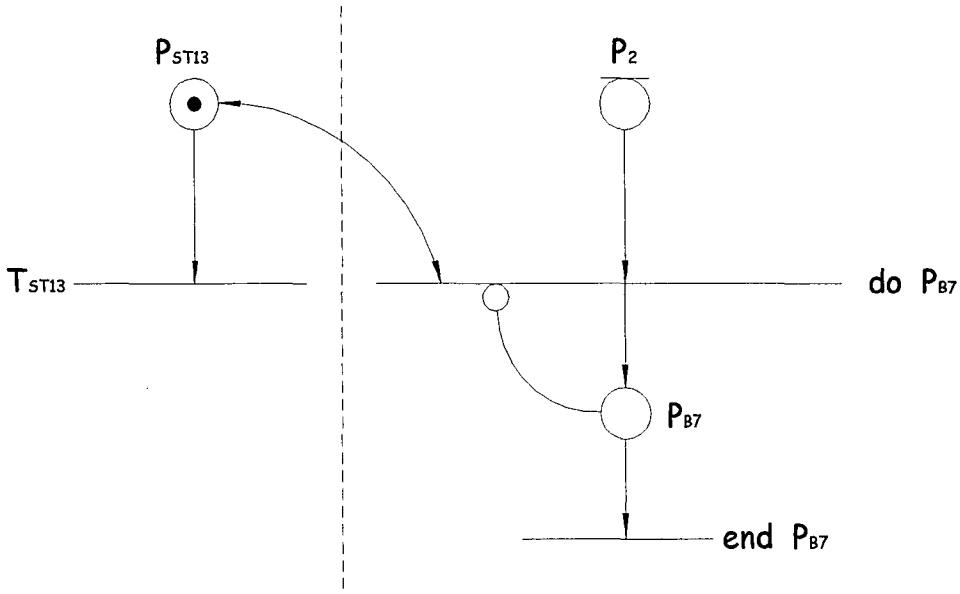
$P_{B6} = P_1 \times P_2 \times P_8$



Description: (the PUMA unloads part B from the conveyor (RP_{PUCB}) when conveyor is in the starting position (S_A), the PUMA is in the at-home position (S_{RHP}, and the vision system recognizes part B S_{VB}).

13 $RP_{PRB} = S_{RHP}$

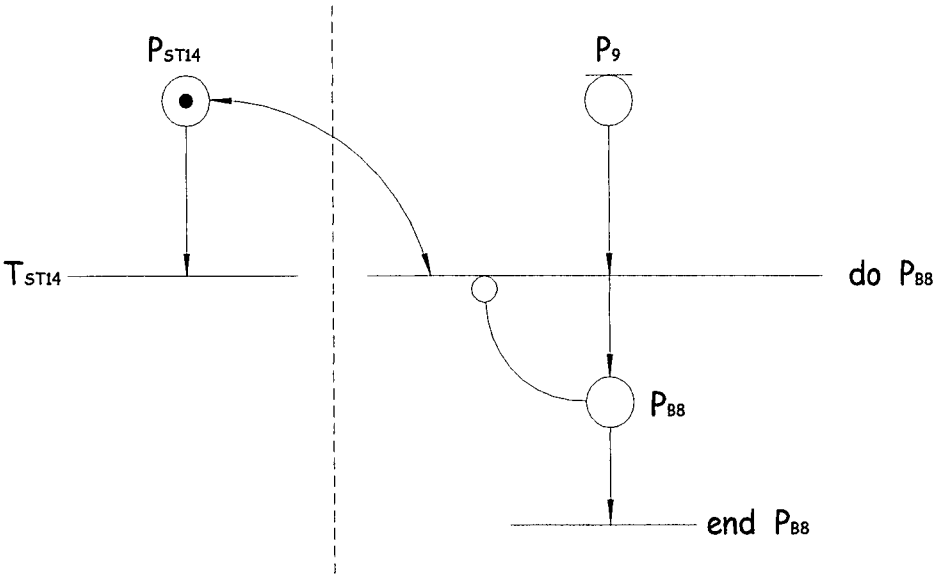
$P_{B7} = P_2$



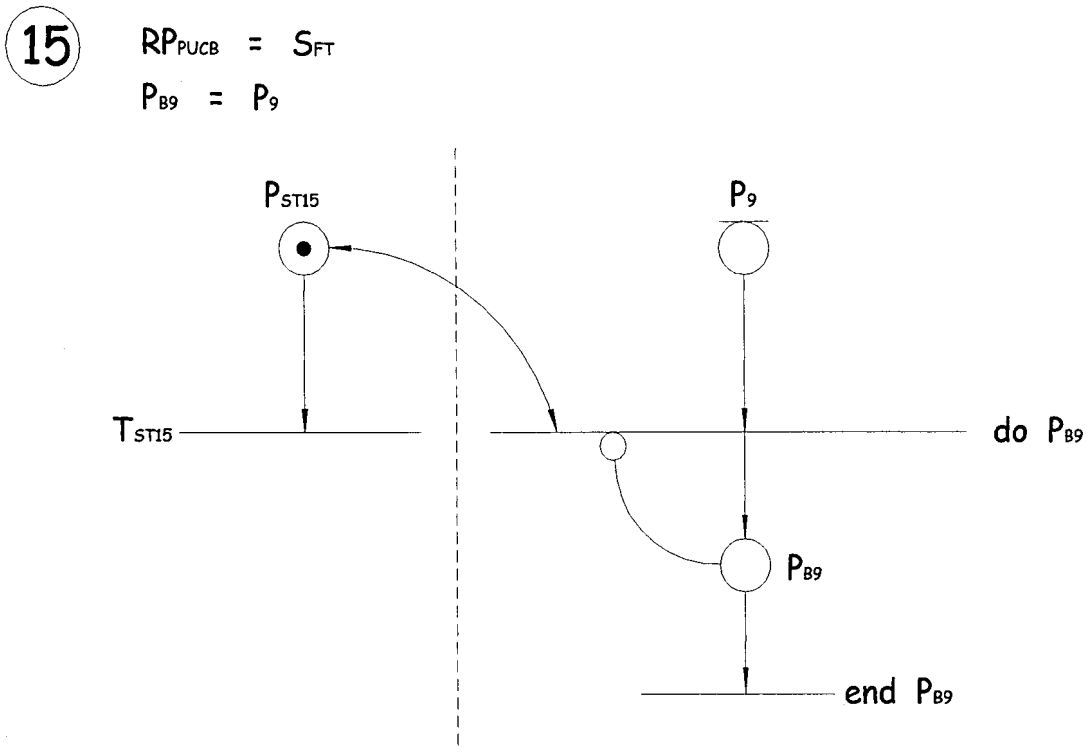
Description: the PUMA retrieves a bolt (RP_{PRB}) when the PUMA is in the at-home position (S_{RHP}).

14 $RP_{PFT} = S_{FT}$

$P_{B8} = P_9$



Description: the PUMA inserts the bolt into part B (RP_{PFT}) when the force-torque sensor has not been activated (S_{FT}).



Description: the PUMA removes part B and places it in storage (RP_{PUCB}) when the force-torque sensor has been activated (S_{FT}).

Vita

Michael J. Smith was born in Scranton, PA in 1979 to James and Cheryl Smith. He received his Bachelor of Science Degree in Industrial Engineering from Lehigh University in January of 2002. Upon completion of the undergraduate program, he entered graduate school. He will be receiving his Master of Science Degree in Industrial Engineering, also from Lehigh University, in May of 2003. Upon graduation, he will be taking a position as a Manufacturing Engineer with Boeing Integrated Defense Systems in Ridley Park, PA.

END OF TITLE