

Computer Science and Systems Analysis
Computer Science and Systems Analysis
Technical Reports

Miami University

Year 1995

Investigation of Programming Languages
for an Automated Manufacturing System

Mark Ma

Miami University, commons-admin@lib.muohio.edu



MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1995-005

**Investigation of Programming Languages for an
Automated Manufacturing System
Mark H. Ma**



School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

Investigation of Programming Languages for an
Automated Manufacturing System

by

Mark H. Ma
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #95-005

July, 1995

Investigation of Programming Languages for an Automated Manufacturing System

Final Report

by Mark H. Ma

Department of Systems Analysis

Miami University

Oxford, Ohio

August, 1995

**Submitted to the faculty of Miami University
in Partial Fulfillment of the requirements for the degree
of Master of Systems Analysis**

Committee Members

Professor Douglas Troy	Advisor : _____
Dr. Alton F. Sanders	Member : _____
Dr. Donald L. Byrkett	Member : _____

ABSTRACT	III
1. INTRODUCTION	1
2. LANGUAGES FOR CONTROL PROGRAMMING	3
2.1 INTRODUCTION TO PROGRAMMING LANGUAGES FOR CONTROL	3
2.1.1 <i>Textural Languages</i>	3
2.1.2 <i>Graphic Languages</i>	5
2.2 DESCRIPTION OF LADDER LOGIC	8
3. CPL LANGUAGE	13
4. LADDER LOGIC ENVIRONMENT	17
4.1 INTRODUCTION	17
4.2 OMEGA CONTROLWARE	17
5. IMPLEMENTING OMEGA CONTROLWARE.....	19
5.1 OC APPLICATION	19
5.1.1 <i>Data Tables</i>	19
5.1.2 <i>Relay Ladder Logic Code</i>	21
5.1.3 <i>Processes</i>	24
5.2 RLL CODE TO CONTROL THE MANUFACTURING CELL	25
5.2.1 <i>Overview</i>	25
5.2.2 <i>RLL Code</i>	26
5.3 INTERFACING PROCESSES.....	43
5.3.1 <i>Polling the Acquisition Board</i>	44
5.3.2 <i>Communication with Robot and Lathe</i>	47
5.3.3 <i>Robot Software Interface</i>	48
5.3.4 <i>CNC Machine Software Interface</i>	50
5.3.5 <i>Communication with Automated Storage and Retrieval System</i>	54
6. COMPARISON OF LADDER LOGIC LANGUAGE VS CPL	58
6.1 STRENGTHS OF RLL OVER CPL	58
6.2 WEAKNESSES OF RLL USING OC	59
7. CONCLUSION.....	60
REFERENCES	62
APPENDIX A COMPLETE RLL PROGRAM USING OMEGA CONTROWARE	63

Abstract

This paper is an investigation of alternative programming languages for use in manufacturing control applications. After reviewing several types of languages, two alternative languages for programming the flexible manufacturing cell in Miami University's Manufacturing Engineering Department are investigated. One language, called Cell Programming Language (CPL), is an object-like high level language developed at Miami University. The other is Relay Ladder Logic (RLL) which is the predominant language used in industry to program programmable logic controllers. An RLL program that is equivalent to an existing CPL program was developed for this purpose.

1. Introduction

A flexible manufacturing system (FMS) can be considered as a set of work cells that operate and are scheduled independently of each other [Benhabib89]. Each individual work cell is composed of one or more machine tools linked by a common material handling system and under the control of a centralized work cell controller for the purpose of producing the given requirements of a family of parts [Martin89]. The work cell controller is programmed to coordinate the interoperation of the various devices in the workcell.

A typical FMS work cell may consist of robots, conveyors, CNC machines, sensors, and other devices. Devices are connected to the cell controller computer such as a PC or Programmable Logic Controller (PLC) through some interface electronics and data acquisition boards. The interface electronics serves to convert signals from the PC or PLC to appropriate signals for these devices. Some devices, such as robots and CNC machines are controlled by programs written in the host command languages of these machines.

An example of a workcell is in the Manufacturing Engineering Department's CIM Laboratory at Miami University, and consists of two robots, one CNC lathe, one conveyor, and one automatic storage system. The relevant sensors are wired, through an external relay interface, to a data acquisition board in a PC. The robots, lathe, and storage system are connected to either the serial port or parallel port on the PC.

Cell Programming Language (CPL) is an object-like workcell programming language developed at Miami University for use by students in the Manufacturing Engineering Department for programming the PC that controls the work cell [Meghamala92]. Currently, CPL is used only at Miami.

The purpose of this project is to evaluate the feasibility of using Relay Logic Language (RLL) as an alternative to CPL for programming the PC controller of the FMS work cell in the Manufacturing Engineering CIM Laboratory at Miami University. RLL code will be developed to control the cell and RLL will be compared to Cell Programming Language (CPL).

Section 2 will review languages for programming work cells. Section 3 describes the CPL work cell programming language. Section 4 presents an overview of relay ladder logic. Section 5 describes an implementation of RLL code to control a work cell. Section 6 presents a comparison between RLL and CPL, and section 7 concludes this report.

2. Languages for Control Programming

2.1 Introduction to programming languages for control

A program for manufacturing control is defined as a “logical assembly of all the programming language elements and constructs necessary for the intended signal processing required for the control of a machine or process by a programmable controller system” [IEC 1131-1].

The International Electrotechnical Commission (IEC) has proposed a set of languages for writing control programs [IEC 1131-1]. They can be categorized as textual or graphic. The languages in each category are described below.

2.1.1 Textual Languages

The two textual languages defined in the standard are called IL (Instruction List) and ST (Structured Text).

Instruction list programs are composed of a sequence of low-level instructions, similar to assembly language. Each instruction begins on a new line and contains an operator with optional modifiers, and, if necessary for the particular operation, one or more operands separated by commas. Operands can be either literals or variables.

An example of the instruction list code is shown in the Table 1.

Label	Operator	Operand	Comment
START:	LD	%IX1	(* PUSH BUTTON *)
	ANDN	%MX5	(* NOT INHIBITED *)
	ST	%QX2	(* FAN ON *)

Table 1. Example of the instruction list code

The semantics of operator LD is to set the current result equal to operand; the semantics of operator ANDN is Boolean AND NOT; the semantics of operator ST is to store the current result to operand location.

The instruction LD %IX1 is interpreted as

result := %IX1

The instruction ANDN %MX5 is interpreted as

result := result AND NOT %MX5

The instruction ST %QX2 is interpreted as

%QX2 := result

A structured text program is composed of high-level statements and expressions, similar to a third-generation high level programming language. An expression is a construct, which, when evaluated, yields a value. Expressions are composed of operators and operands. An operand can be literals, variables, function invocations, or another expression. The operators of the ST language are summarized in the Table 2.

No.	Operation	Symbol	Precedence
1	Parenthesization	(expression)	HIGHEST
2	Function evaluation	identifier(argument list)	
3	Exponentiation	**	
4	Negation	-	
5	Complement	NOT	
6	Multiply	*	
7	Divide	/	
8	Module	MOD	
9	Add	+	
10	Subtract	-	
11	Comparison	<, >, <=, >=	
12	Equality	=	
13	Inequality	<>	
14	Boolean AND	&	
15	Boolean AND	AND	
16	Boolean Exclusive OR	XOR	
17	Boolean OR	OR	LOWEST

Table 2. Operators of the ST language.

The statements of the ST language are summarized in Table 3. Statements are terminated by semicolons.

An equivalent example (see Table 1) of the Structured Text is showed below:

```
a := b;  
a := a AND NOT c;  
d := a;
```

No.	Statement type/Reference	Examples
1	Assignment	A:=B; CV:=CV+1;C:=SIN(X);
2	Function block Invocation and FB output usage	CMD_TMR(IN:=%IX5, PT:=T#300MS); A:=CMD_TMR.Q;
3	RETURN	RETURN;
4	IF	D:=B*B-4*A*C; IF D < 0.0 THEN NROOTS:=0; ELSEIF D := 0.0 THEN NROOTS := 1; X1 := -B/(2.0*A); ELSE NROOTS := 2; X1 := (-B+SQRT(D))/2.0*A); X2 := (-B-SQRT(D))/2.0*A); END_IF;
5	CASE	TW:=BCD_TO_INT(THUMBWHEEL); TW_ERROR :=0; CASE TW OF 1,5 : DISPLAY:= OVEN_TEMP; 2 : DISPLAY := MOTOR_SPEED; 3 : DISPLAY := GROSS-TARE; 4,6..10 : DISPLAY := STATUS(TW-4); END_CASE; QW100 := INT_TO_BCD(DISPLAY);
6	FOR	J := 101; FOR I := 1 TO 100 BY 2 DO IF WORDS[I] = 'KEY' THEN J := I; EXIT; END_IF END_FOR;
7	WHILE	J := 1; WHILE J <= 100 & WORDS[J] <> 'KEY' DO J := J+2; END_WHILE;
8	REPEAT	J := -1; REPEAT J:= J+2; UNTIL J=101 OR WORDS[J] = 'KEY' END_REPEAT;
9	EXIT	EXIT;
10	EMPTY STATEMENT	::

Table 3. ST languages statements

2.1.2 Graphic Languages

The two graphic languages defined in the standard are LD (Ladder Diagram), also called Relay Ladder Logic (RLL), and FBD (Function Block Diagram) [IEC1131-1].

Link elements may be horizontal or vertical. The state of a link element can be either “ON” or “OFF”, corresponding to the literal Boolean values 1 or 0, respectively. A horizontal link element is indicated by a horizontal line. A horizontal link element transmits the state of the element on its immediate left to the element on its immediate right. The vertical link element consists of a vertical line intersecting with one or more horizontal link elements on each side. The state of the vertical link represents the inclusive OR of the ON states of the horizontal links on its left side.

A **contact** is an element which imparts a state to the horizontal link on its right side which is equal to the Boolean AND of the state of the horizontal link at its left side with an appropriate function of an associated Boolean input, output, or memory variable. A contact does not modify the value of the associated Boolean variable.

A **coil** copies the state of the link on its left to the right without modification, and stores an appropriate function of the state or transition of the left link into the associated Boolean variable.

Within a program organization unit written in LD, networks are evaluated in top to bottom order as they appear in the ladder diagram, except as this order is modified by the execution control elements.

The other graphic language is Function Block Diagram (FBD). FBD represents **signal flow** which is analogous to the flow of signals between elements of a signal processing system. Signals in the FBD language flow from the output (right-hand) side of a function or function block to the input (left-hand) side of the function or function block(s) so connected. An example of a FBD is shown in Figure 2.

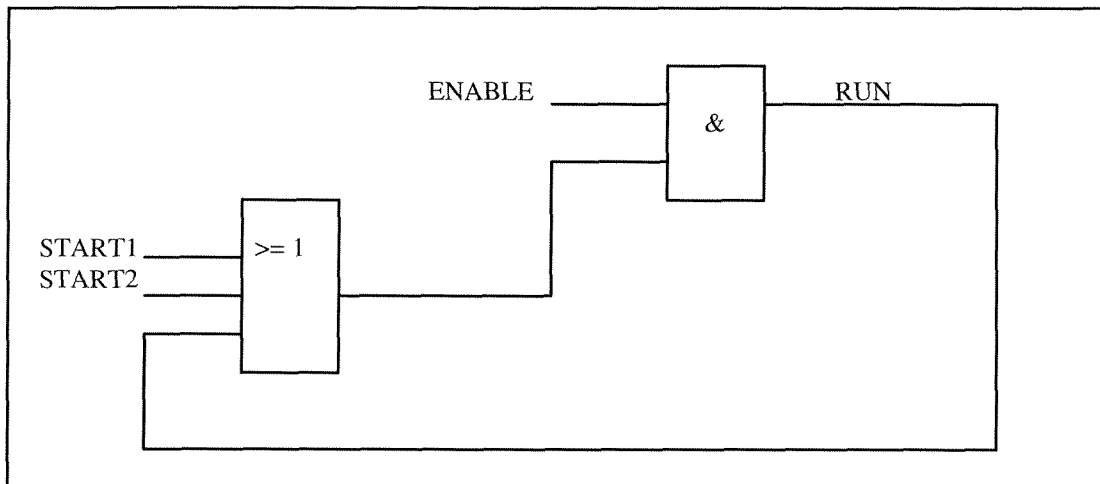


Figure 2 Feedback path example using function block diagram

Within a program organization unit written in the FBD language, the order of network evaluation follows the rule that the evaluation of a network must be complete before starting the evaluation of another network which uses one or more of the outputs of the preceding evaluated network.

Since the most common programming language for controllers is ladder logic [Chaar90], this language is described in more detail in the next section.

2.2 Description of Ladder Logic

Ladder diagrams, which have been used for decades for describing relay circuits, are now being utilized for programming programmable logic controllers (PLCs). This is so because many practicing engineers and technicians are familiar with these diagrams, and feel comfortable working with them [Pessen89]. Ladder diagrams were thus adapted as a graphical programming language for PLCs.

Ladder logic programming typically deals with relays (contacts) and coils. Contacts can be 'Normally Open' (denoted by the ---|/--- symbol) or 'Normally Closed' (denoted by the ---|/|--- symbol). 'Coils' are denoted by the ---()--- symbol. A Normally open contact does not pass power unless its associated coil is energized by applying power to it. A Normally closed contact passes power until its associated coil is energized

by applying power to it. Using these contacts and coils a number of useful circuits can be 'wired' using ladder logic. An example is a 'one shot'. One shot can be used to initialize routines, to turn on outputs or control relays for 1 scan using a switch. Figure 3 shows an example of a one shot used to turn on control relay 100 for 1 scan when switch 2000 is pressed.

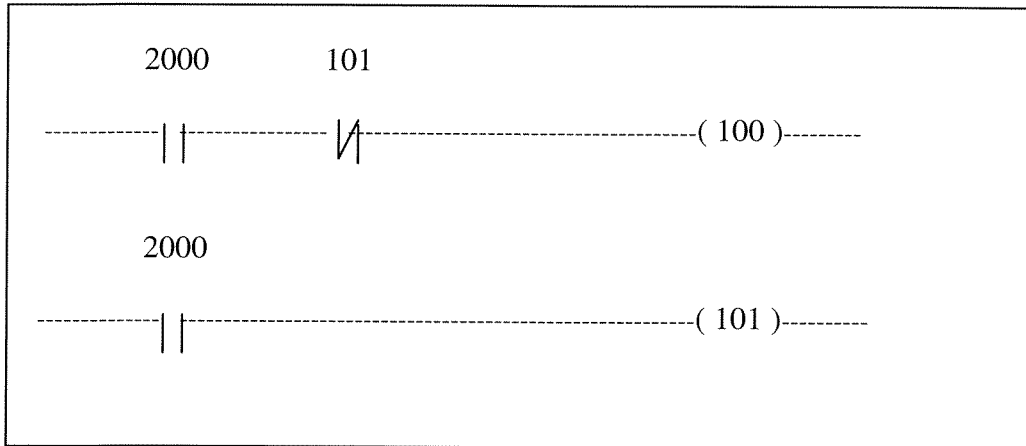


Figure 3 Example of one shot

A Latched relay is another useful device. This relay can latch the state of an input during a scan and can be then used to drive other logic over successive scans. One use can be in recognizing when a proximity switch has been tripped. Figure 4 shows that output 100 becomes energized when contract 2000 is tripped and then stays on forever or until contract 101 is tripped.

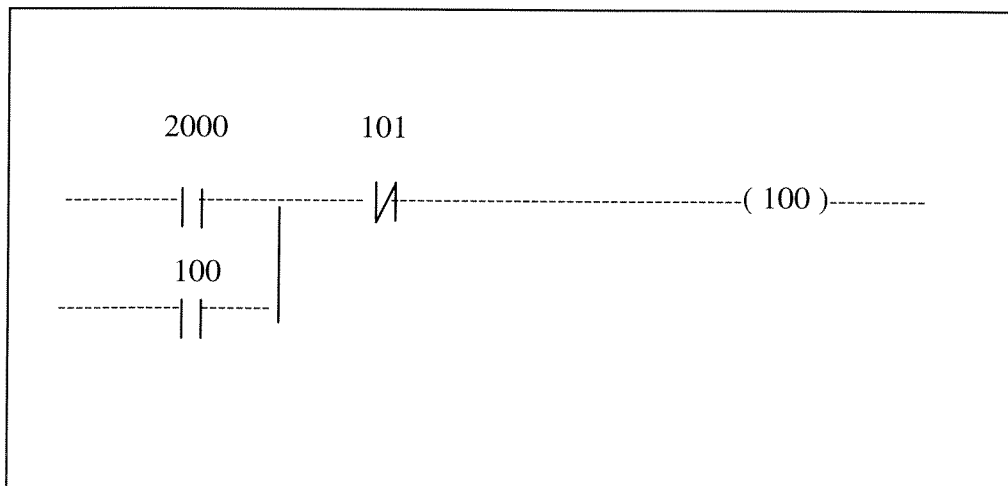


Figure 4 Example of latched relay

A Flip-Flop is a device that changes its output state each time an input is energized. Deenergizing the input has no effect on the output. In figure 5 Control 200 is the output. On the first energization of input 2000, 200 turns on. Note that control relay 100 is a one shot of input 2000.

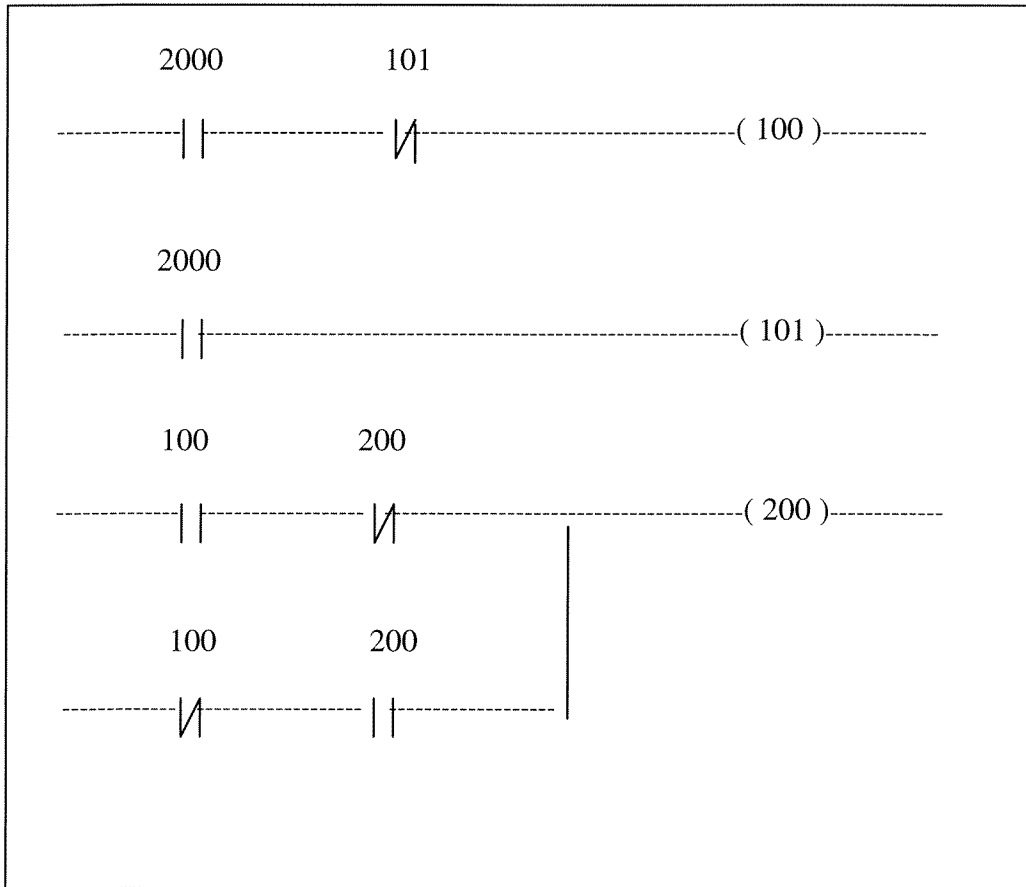


Figure 5 Example of a Flip-Flop

A small ladder program that corresponds to CPL program (turn on the conveyor and wait for pallet to arrive, then turn the conveyor off) is shown below in Figures 6 and 7.

In Figure 6, there are two conditions to turn the conveyor on: start which is set when the lathe process completes, and p_liftdw which is set when pallet lift is down.

In Figure 7, there are also two conditions to turn conveyor off: p_arr which is set when a pallet arrives at one specific position, and t7_out which is set when the whole process completes.

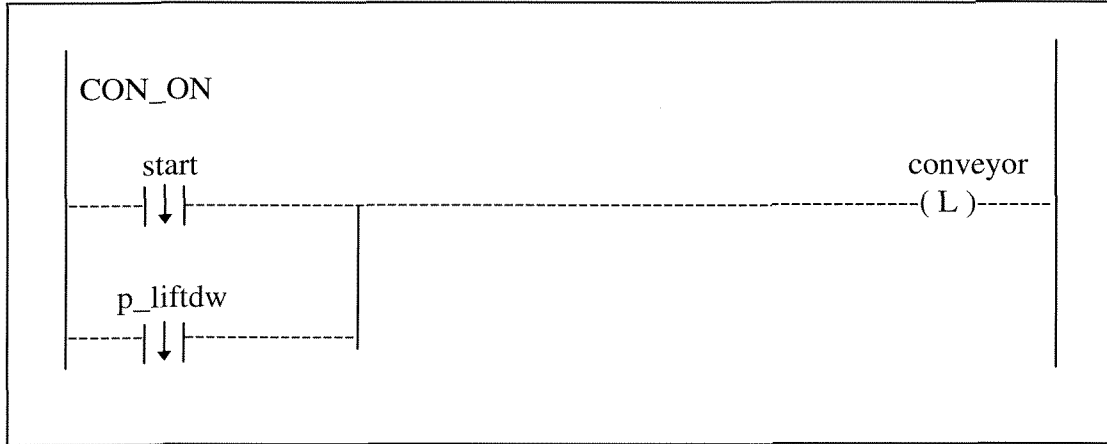


Figure 6 Turn on conveyor

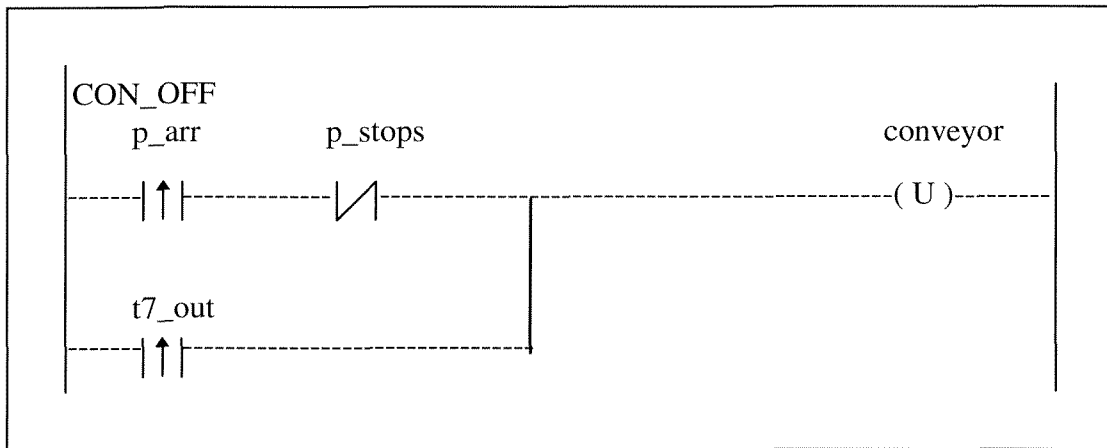


Figure 7 Turn off conveyor

While manufacturers of PLCs usually provide detailed manuals explaining how to enter a given ladder diagram into the controller's memory, these manuals are generally of little or no help in designing the ladder diagram to begin with.

In the past, most ladder diagrams were designed by intuition, for example [Saake70], shows ladder diagrams for various applications, but does not really explain

how these were designed. The method usually employed is to add each element as needed for the next step of the control sequence, and then check to make sure that the element just added does not interfere with previous steps. If it does, another element is added to “patch things up”. It is obvious that the result obtained with such methods depends to a great deal on the designer’s skill and experience. Thus, although RLL is currently the most popular language for programming controllers, it is not clear that it is the best. To evaluate the feasibility of using RLL to program Miami’s FMS cell, RLL is compared with the language currently used for control programming called CPL. CPL language is described in the next section.

3. CPL Language

Cell Programming Language (CPL) is an object-like workcell programming language developed at Miami University for use by students in the Manufacturing Engineering Department [Meghamala92].

An example of a CPL program is shown in Figure 8:

```
Ports
  ComPort      COM1      300 7 2 0;
  PortA        64256     Input;
  PortB        64257     Output;
  PortC        64259     Output;
End

Device
  Conveyor     Coil           PortC      5;
  PhotoCell    Sensor        PortA      7;
  Robot        Programmable  LPT1;
  Lathe        Programmable  ComPort;
  Delay        Wait;
End

Procedure
  Lathe.Do(loadlath);
  Robot.Send("NT");
  Robot.Do(loadpart);
  Conveyor.On;
  PhotoCell.WaitOn;
  Delay.500;
  Conveyor.Off;
End
```

Figure 8 An example of a CPL program.

A CPL program consists of three sections : port declarations, device declarations, and the procedure declaration.

The port declaration section is used to assign physical I/O port addresses to each register on the data acquisition board in the PC and to define data flow direction (input/output) of each register. In CPL these addresses are given port names for later reference. Port name can also be assigned to serial (COM) ports or parallel (LPT) ports.

The device declaration section is used to declare device objects and associate a port and a bit number with each device object. The device types are predefined in the

language. Each declaration consists of a device_identifier, a device_type, and a port_identifier and bit_number.

The device_identifier is a user defined name and device_type is keyword defined in the CPL language. The device_types are shown in the Table 4. The port_identifier should be a name defined in the port declaration section, and the bit_number is a constant between 0 and 7 and corresponds to a bit on the data acquisition board. For a programmable device type, the port name COM1, COM2, or LPT1 would be specified depending on which communication port is connected to the device. Each device_type has a set of functions (or methods) shown in Table 4 that can be used with a device of that type in the procedure section.

TYPES	VALID FUNCTION
COIL	ON, OFF
SENSOR	WAITON, WAITOFF
PULSE	STROBE
PROGRAMMABLE	SEND, DO
DELAY	MILLISECONDS

Table 4. Device Types and Valid Function

The procedure section consists of control statements. Each statement represents one device operation and directly corresponds to an actual operation of the real device in the FMS cell. There is only one procedure section and all statements are executed in sequence. There are no control constructs, such as loops or conditions, and no subroutines.

To completely control a FMS cell, a complete CPL-based program requires robot programs and CNC programs as well as the CPL program. An example of a complete program to control the cell, that is in use at Miami, is shown in Figure 9.

```

Ports          /* Port declarations
    PortC          64259      Output;
    PortA          64256      Input;
End

Devices        /* Device declarations
    PalletLiftUp   Pulse      PortC  4;
    Conveyor       Coil       PortC  5;
    PhotoCell      Sensor     PortA  7;
    PalletArrived  Sensor     PortA  6;
    ChuckOpen      Pulse      PortC  1;
    Lathe          Programmable COM1;
    Robot          Programmable COM2;
    LatheStart     Pulse      PortC  2;
    LatheStop      Sensor     PortA  4;
    PalletLifted   Sensor     PortA  5;
    PalletStops    Coil       PortC  0;
    ChuckClose     Pulse      PortC  3;
    PalletLiftDown Pulse      PortC  6;
    LatheRunning   Sensor     PortA  2;
    LatheHandShk   Sensor     PortA  3;
    Delay          Wait;
End

Procedure      /* Device operations
1    Lathe.Do(loadlath);
2    Robot.Send("NT");
3    PalletStops.On;
4    Conveyor.On;
5    PhotoCell.WaitOn;
6    PalletStops.Off;
7    PalletArrived.WaitOn;
8    Delay.1000;
9    PalletLiftUp.Strobe;
10   PalletLifted.WaitOn;
11   Conveyor.Off;
12   ChuckOpen.Strobe;
13   Robot.Do(LoadPart);
14   Delay.1000;
15   ChuckClose.Strobe;
16   Delay.2000;
17   Robot.Do(MoveAway);
18   Delay.2000;
19   LatheStart.Strobe;
20   LatheStop.WaitOff;
21   Robot.Do(MoveBack);
22   Delay.2000;
23   ChuckOpen.Strobe;
24   Delay.2000;
25   Robot.Do(GetPart);
26   PalletStops.On;
27   PalletLiftDown.Strobe;
28   Conveyor.On;
29   Delay.500;
30   Conveyor.Off;

```

```
31   LatheStart.Strobe;  
32   PalletStops.Off;  
End;
```

Figure 9 CIM Lab CPL Program [Meghamala92]

(Note : Numbers are marked in procedure are for reference later in this report. They are not part of the actual program)

CPL does not hide all of the hardware details. In order to use CPL, the user is still required to know the hardware port address and bits to which each device is interfaced. Also, the user must know the type of each device. Finally, individual cell components, such as robots and CNC machine, will have to be programmed in their host languages. In figure 9, the names Loadpart (line 13), MoveAway (line 17), MoveBack (line 21), and GetPart (line 25) are files containing robot or CNC programs. The contents and creation of these files are described in [Liwu94].

4. Ladder Logic Environment

4.1 Introduction

The primary purpose of this project is to compare RLL to CPL in order to determine the feasibility of using RLL to program Miami's FMS cell. To do so, a RLL programming environment was acquired that allows the cell controller PC to be programmed using RLL. This environment is called Omega Controlware and is explained in the next section.

4.2 Omega Controlware

Typically, ladder logic is used to control a PLC. However, in the Manufacturing CIM Lab, a PC is used to control the manufacturing cell. Omega Controlware is a software product that is used to allow a PC to perform the functions of a PLC [Omega903-19992-1.00R]. Omega also supports programming of the PC using ladder logic. Thus, Omega allows the use of the PC as a ladder logic-based programmed controller.

Omega Controlware (OC) is a preemptive multitasking operating system that operates within a DOS environment and addresses the requirements of machine control, process control, and cell environments. OC support operates on IBM PC or compatible based hardware, including alternate bus-based systems.

PRO, a component of the Omega product, is an editor for the development of Omega Controlware application programs using ladder logic. It offers a mouse-driven interface that utilizes pull-down menus, dialog boxes, scroll bars, and multiple windows, making application development fast and easy.

OC combines the features of some of the more popular relay ladder processor with some unique facilities. For example, OC supports networking capabilities, multitasking, and a wide range of data item types [Omega903-19992-1.00R].

Unlike a PLC, Omega Controlware is a set of software development tools, which supports ladder logic programming. To use Omega Controlware, the host PC needs some hardware support to interface with the control environment, such as a data acquisition

board, relay isolated output electronics, and optical isolated input electronics. Also, some additional coding of a software interface between Omega Controlware and this hardware, which is called process by Omega, is required. This will be explained below.

Figure 10 gives an overview of the various components that make up an OC application.

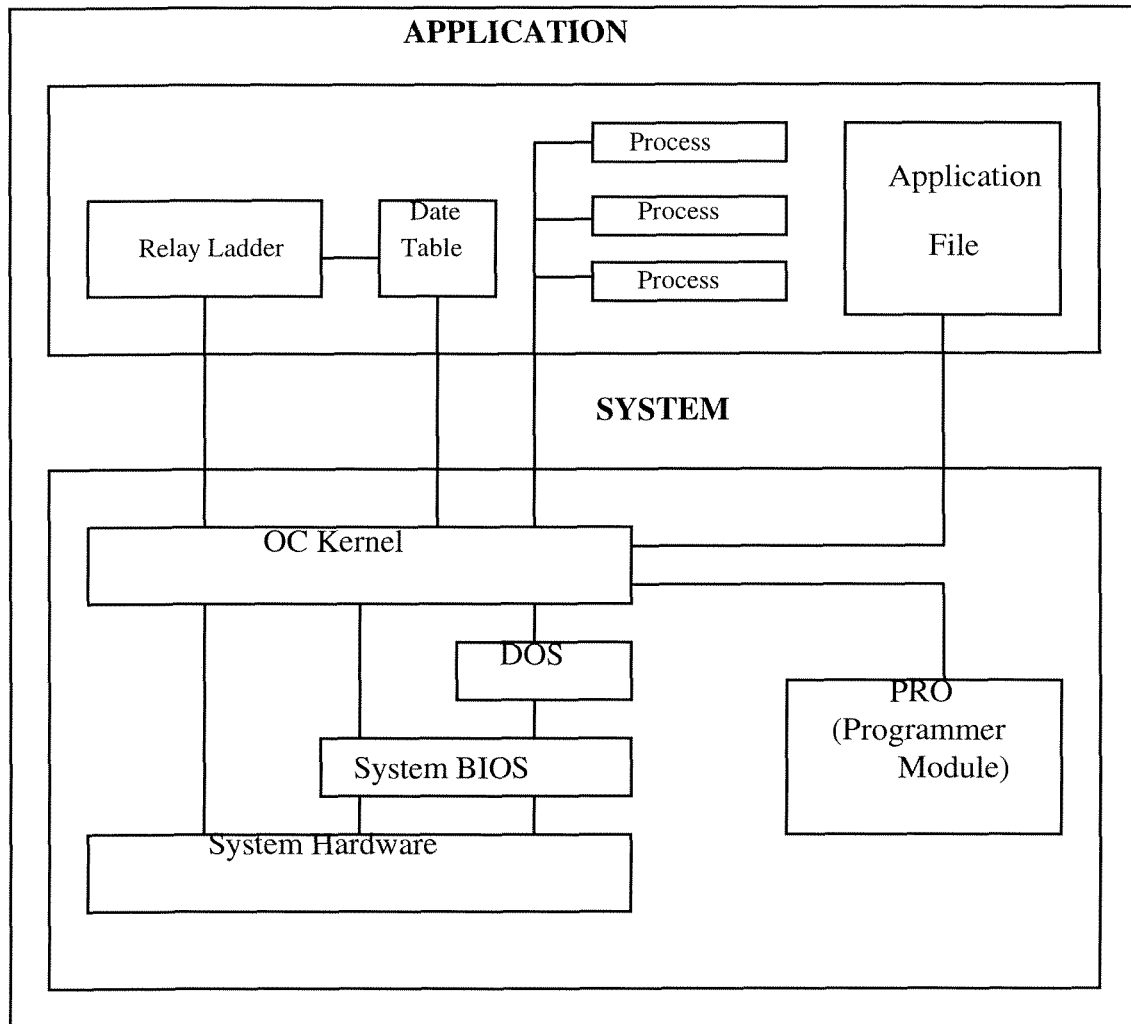


Figure 10 Overview of an OC Application

PRO is a state of the art editor for the development of Omega Controlware application programs. It offers a mouse-driven interface that utilizes pull-down menus, dialog boxes, scroll bars, and multiple windows, making application development fast and easy. Relay Ladder, Data Table and Process will be explained below.

5. Implementing Omega Controlware

5.1 OC application

An OC application consists of three parts : Data tables, relay ladder logic code, and processes. These are described below.

5.1.1 Data Tables

The data used by the relay ladder code, and all interprocess communication in OC, is achieved through the use of a data item table, known as the data table. A wide range of data item types are supported by OC [Omega903-19992-1.00R], including integer and real numeric variables, string, groups, semaphores, memory and disk arrays, mailbox, and stack. The total number and size of data items is only limited by available memory and disk space.

Each data item is identified and referenced by a one to eight character name.

The data table corresponding to the example CPL program from Figure 9 is shown below in Table 5.

Data Type	Name	Alias	Description
Ubyte	port_a	(CIM1)	' CIM Cell Control Input Model '
	.4	(La_Stop)	' Lathe Stop '
	.5	(P_Lifted)	' Pallet Lifted '
	.6	(P_Arr)	' Pallet Arrived '
	.7	(Pho_Cell)	' Photo Cell '
Ubyte	port_b	(CIM2)	' CIM Cell Control Output Model '
	.0	(La_Hand)	' Lathe Handshank '
	.1	(La_G66)	' Lathe G66inp '
	.2	(La_Run)	' Lathe Running '
Ubyte	port_c	(CIM3)	' CIM Cell Control Output Module '
	.0	(P_stops)	' Pallet Stops '
	.1	(Ch_open)	' Chuck Open '
	.2	(La_start)	' Lathe Start '
	.3	(Ch_close)	' Chuck Close '
	.4	(P_liftup)	' Pallet Lift Up '
	.5	(Conveyor)	' Conveyor '
	.6	(P_liftdw)	' Pallet Lift Down '
Bit	process2		' Run process robot (nest.cmd) '
Bit	process3		' Run process lathe '
Bit	process4		' Run process poll '
Bit	process5		' Run process robot (loadpart.cmd) '
Bit	process6		' Run process robot (moveaway.cmd) '
Bit	process7		' Run process robot (moveback.cmd) '
Bit	process8		' Run process robot (getpart.cmd) '
Bit	process9		' Run process reset '
Bit	start		' Start conveyor running '
Bit	enable1		' time1 enable '
Bit	enable2		' time2 enable '
Bit	enable3		' time3 enable '
Bit	enable4		' time4 enable '
Bit	enable5		' time5 enable '
Bit	enable6		' time6 enable '
Bit	enable7		' time7 enable '
Bit	t3_out		' time3 output '
Bit	t6_out		' time6 output '
Bit	t7_out		' time7 output '
Word	time1		' Delay 1000 milliseconds then lift Pallet up '
Word	time2		' Delay 1000 milliseconds then get chuck close '
Word	time3		' Delay 2000 milliseconds '
Word	time4		' Delay 2000 milliseconds then start lathe '
Word	time5		' Delay 2000 milliseconds then open chuck '
Word	time6		' Delay 2000 milliseconds '
Word	time7		' Delay 500 milliseconds '

Table 5. Example of data table

5.1.2 Relay Ladder Logic Code

There are four types of items to code in a relay ladder logic application file. These are defined in Table 6.

Title items :	Used to define a title for a application
Data items :	Used to define data storage for use in the application
Rung items :	Used to define the relay ladder portion of the program
Memo items :	Used to place comment text into the program







Table 6. Types of items within an application.

A ladder consists of one or more rungs. The rungs in a ladder are executed from top to bottom, starting with the root ladder which is always the topmost ladder in an application. Every application contains one root ladder and can optionally contain any number of named subladders.

There are a number of contact and coil types that can be used to create rungs in OC. Some are standard to all relay ladder languages while others are unique to Omega Controlware.

Basically, a **contact** tests whether an integral data item contains a zero or non-zero value. The tested data item can be a bit or a signed/unsigned byte, word, or double word. Other data types cannot be tested as contacts. OC allows both horizontal and vertical wires to be entered into contact fields. These wires provide connections between contacts and coils.

The following contact types are supported in OC:

- Normally Open Contact 
- Normally Closed Contact 
- Leading Edge Contact 
- Trailing Edge Contact 
- Not Leading Edge Contact 
- Not Trailing Edge Contact 

The **Timer Rung** counts from a preset value to a limit value by 1 at a specified rate. A timer can have any interval from 1 millisecond to 49.7 days. Each time the rung is executed with the enabling logic true, the system checks to see if the specified interval has elapsed. If so, the timer accumulator is incremented. If the accumulator reaches or exceeds the limit, the LIMIT coil is activated, and the timer is disabled. A 5*4 matrix can be divided as needed to provide both enabling and reset logic for the timer.

The Figure 12 is a sample of Timer Rung .

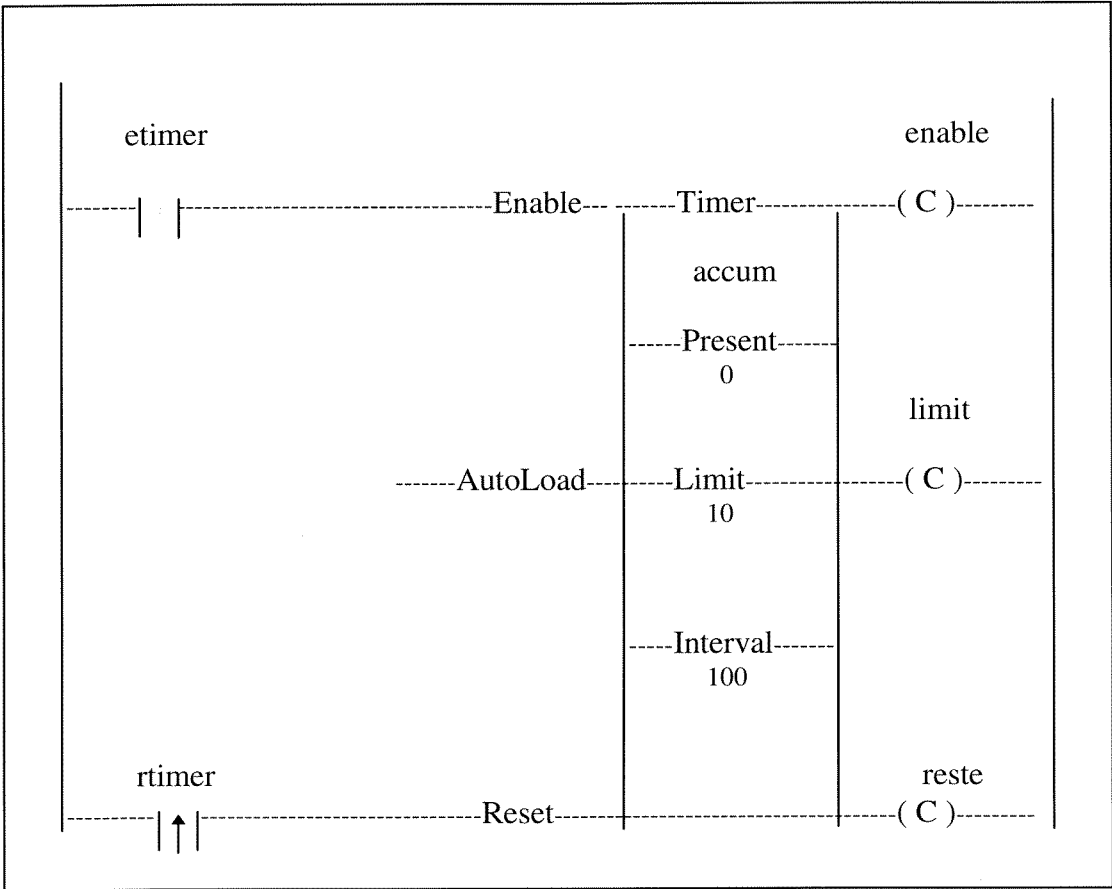


Figure 12 Timer rung

The Queue Rung will queue the named process (.exe file) for later execution if the enabling logic evaluates true. A command line can be passed to the process. The command line is accessed by the processes using system functions or by using the normal high-level language facilities for accessing a DOS command line. A 5*6 matrix can be

used as needed to provide enabling logic for the rung. The Figure 13 is a sample of Queue Rung.

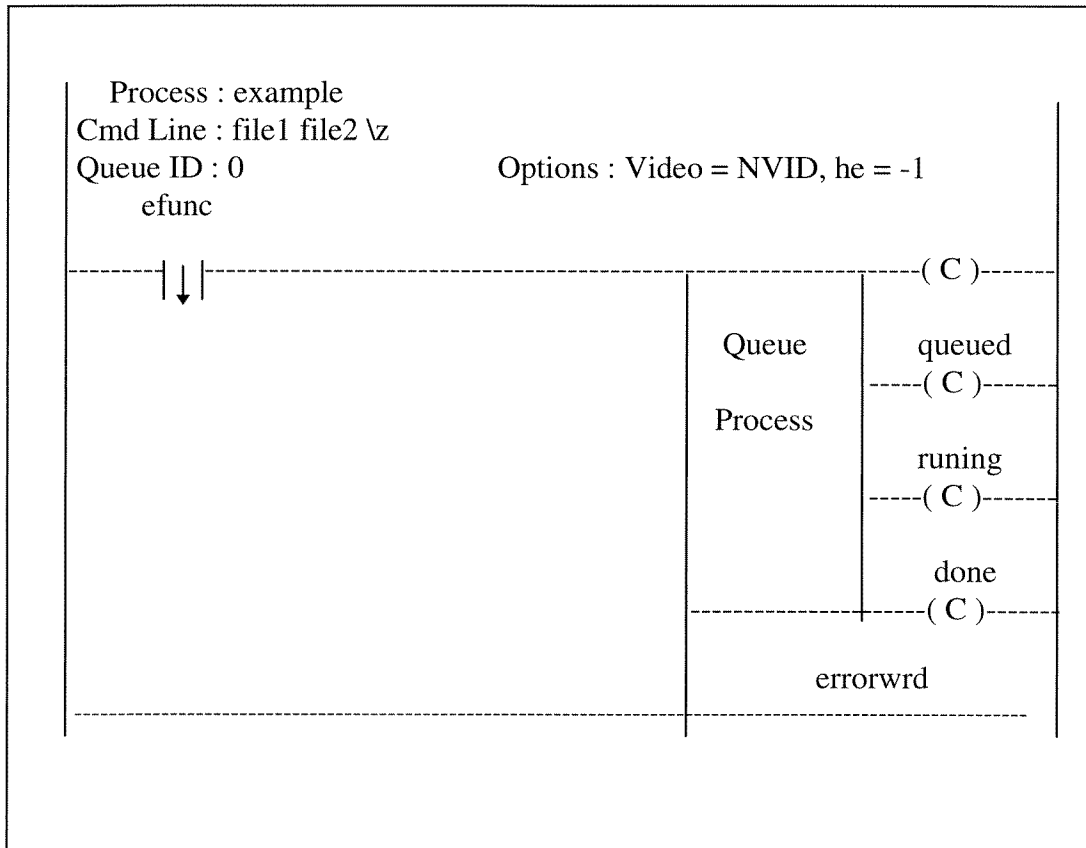


Figure 13 Queue rung

5.1.3 Processes

A process is an instance of an executing program plus all of the resources used by the program. Each process is a DOS executable (.exe) file that is created by either a high-level language compiler or an assembler. A process can be started by a queue rung in the relay ladder or by another process.

When a process is started, it is placed into one of 256 user process queues. Within each queue, processes are executed on a first in, first out (FIFO) basis.

For this project, processes had to be written to interface between OC and the data acquisition board, the robot, and the CNC machine. This problem is described in the next section.

5.2 RLL Code to Control the Manufacturing Cell

5.2.1 Overview

To compare the use of ladder logic using OC to CPL, a ladder logic program that is functionally equivalent to the example CPL program in Figure 9 was developed. Using OC to do this required the development of several modules in addition to the ladder logic program. The architecture of these modules is shown in Figure 14.

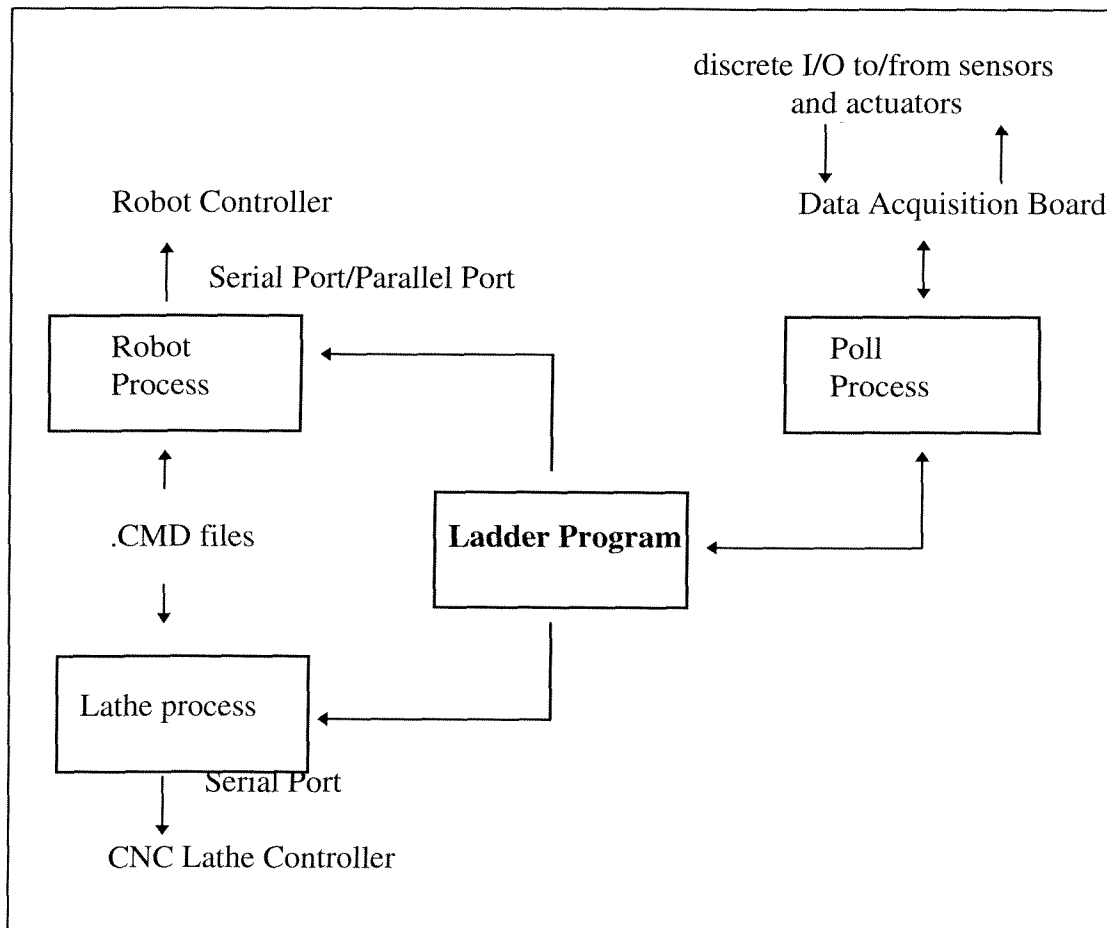


Figure 14 Software architecture using OC

The robot, poll, and lathe processes were written in C using Borland C++. The ladder program was written using the OC PRO ladder logic editor. Each of these are described below.

5.2.2 RLL Code

The complete RLL program that reproduces the functions of the CPL program in Figure 9 is shown in Appendix A. In this section the RLL code that corresponds to each CPL statement (or group of statements) from Figure 9 is described. Each statement is numbered for reference.

Rung1 (CPL statement : 2 Robot.Send("NT");)

This queue rung starts the process called robot that initializes the robot's position by sending the contents of the file nest.cmd to the robot connected to the serial port (Figure 15). The process (robot) is a C program that was written and compiled using Borland C++, and described in section 5.3.3.

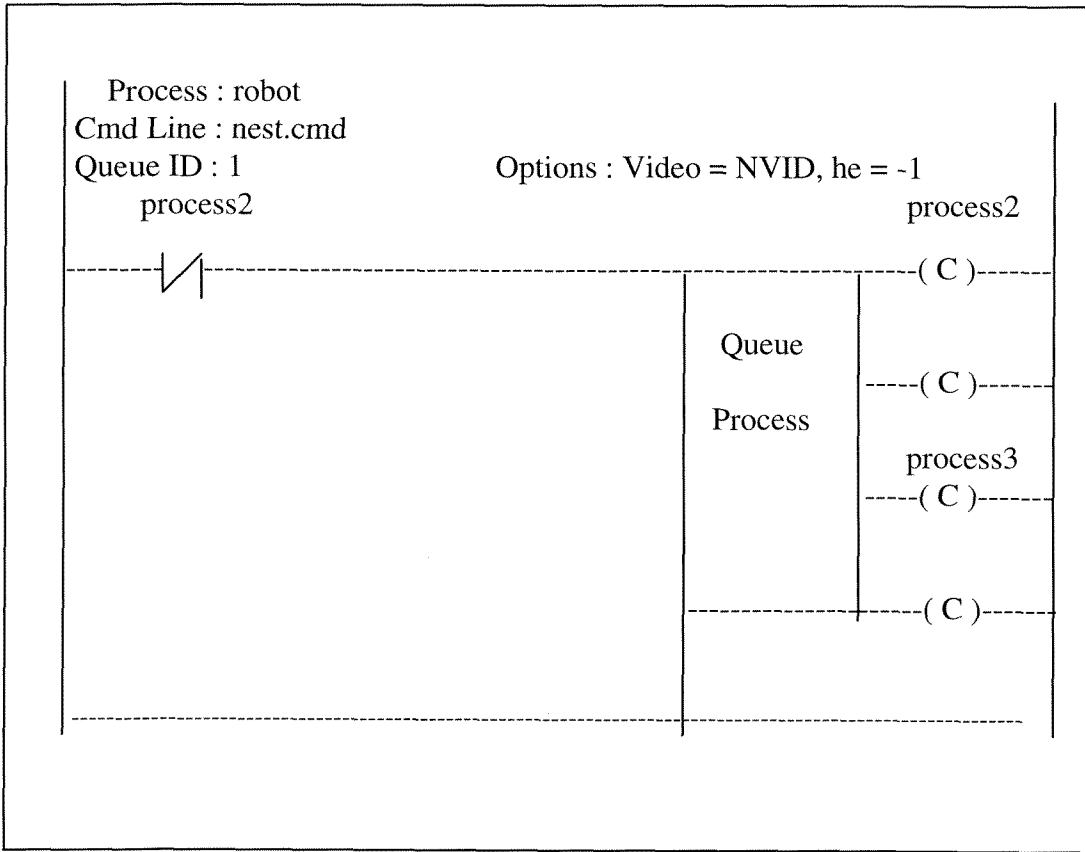


Figure 15 Rung 1

Rung 2 (No corresponding CPL statement)

This queue rung starts the process named poll that polls the acquisition board. Process3 is output from rung1 when the robot process completes. (Figure 16)

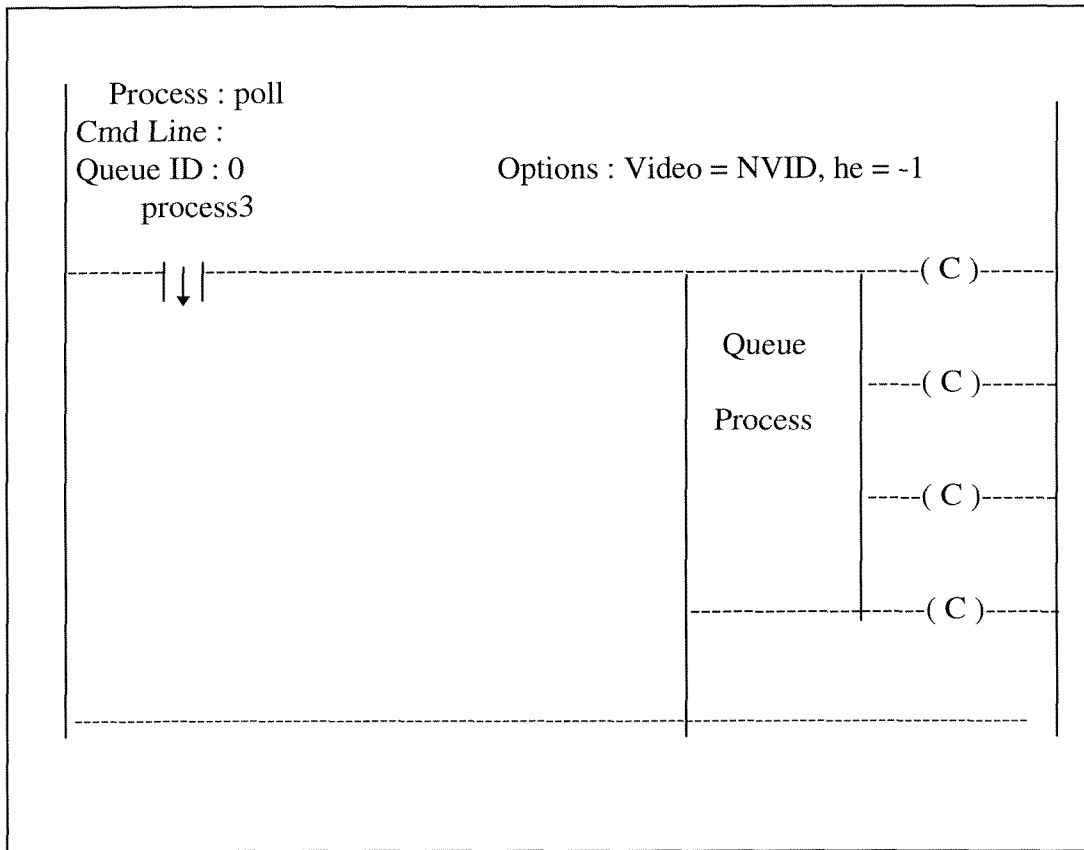


Figure 16 Rung 2

Rung 3 (CPL statement : 1 Lathe.Do(loadlath))

This queue rung starts the process named lathe that loads the lathe program from file lathe.cmd and then starts the conveyor running. Process3 is output from rung1 when the robot process completes. Start is output from this rung when this rung is done (Figure 17). The lathe process is described in section 5.3.4.

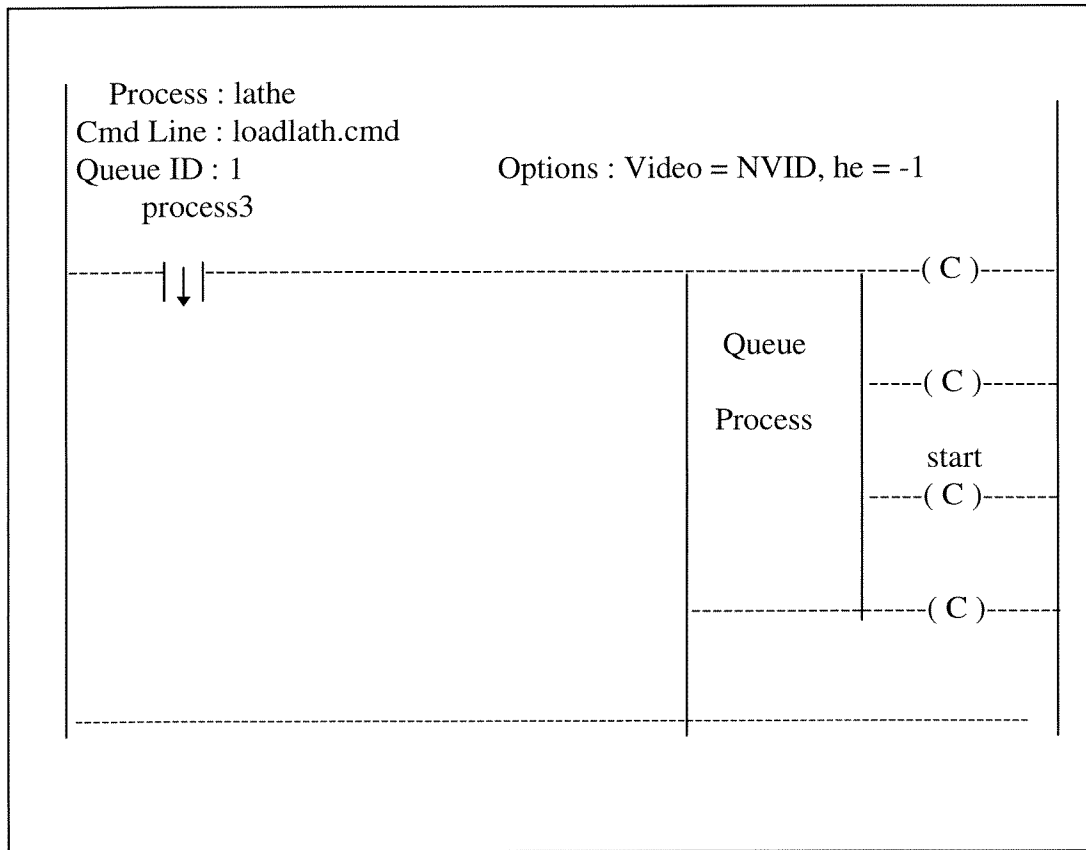


Figure 17 Rung 3

Rung 4 (CPL statement : 7 PalletArrived.WaitOn;
 27 PalletLiftDown.Strobe;
 3, 26 PalletStops.On;)

This matrix rung turns the pallet stops on when the lathe process completes or when pallet is down at the loading position. (Figure 18)

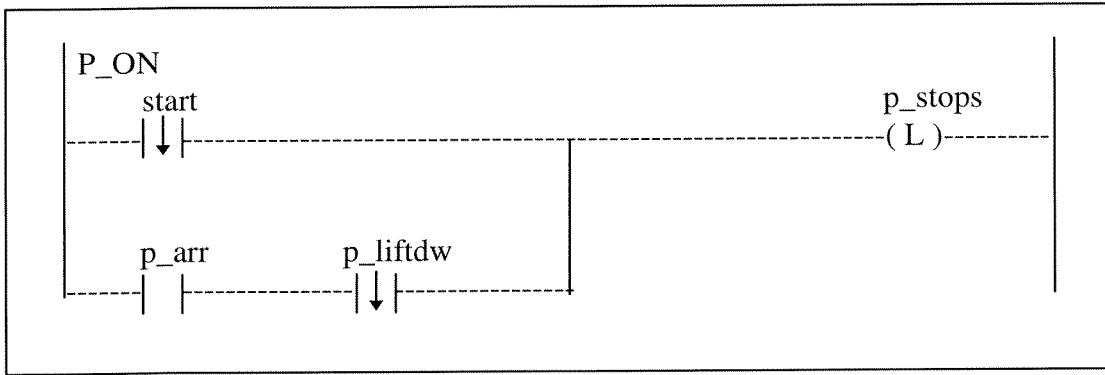


Figure 18 Rung 4

Rung 5 (CPL statement : 5 PhotoCell.WaitOn;
6 PalletStops.Off;)

This matrix rung turns the pallet stops off when the photocell is on or when the whole process completes. (Figure 19)

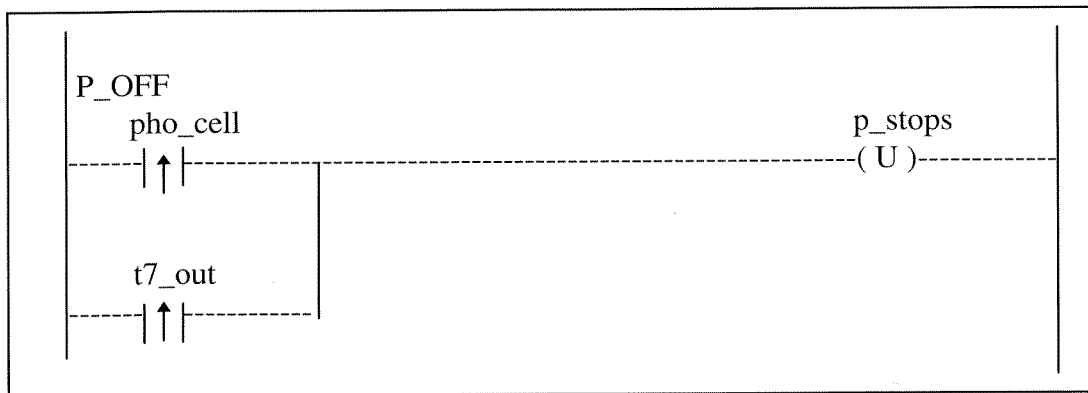


Figure 19 Rung 5

Rung 6 (CPL statement : 27 PalletLiftDown.Strobe;
4 or
28 Conveyor.On;)

This matrix rung turns the conveyor on when the lathe process completes or when the Pallet lift is down at the loading position. (Figure 20)

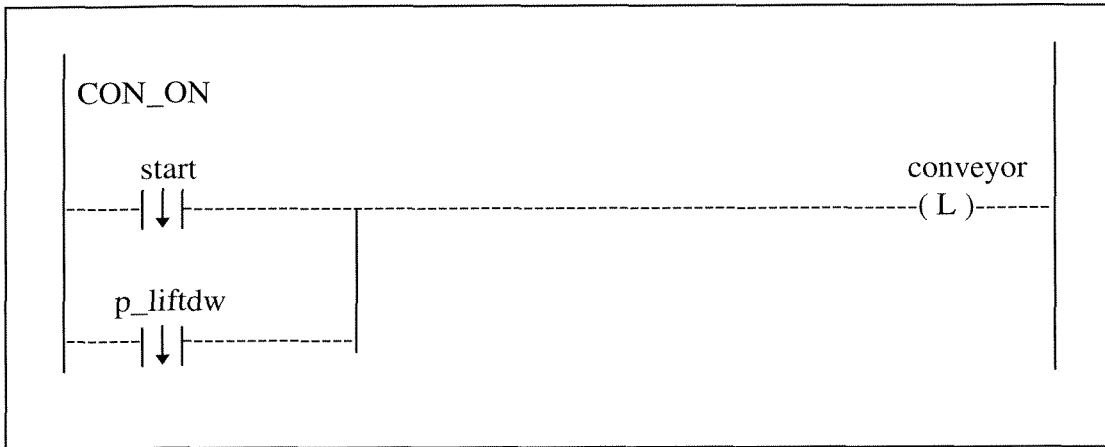


Figure 20 Rung 6

Rung 7 (CPL statement : 7 PalletArrived.WaitOn;
 4 PalletStops.On;
 11 or 30 conveyor.Off;)

This matrix rung turns the conveyor off when a pallet arrives at the loading position or when the whole process completes. (Figure 21)

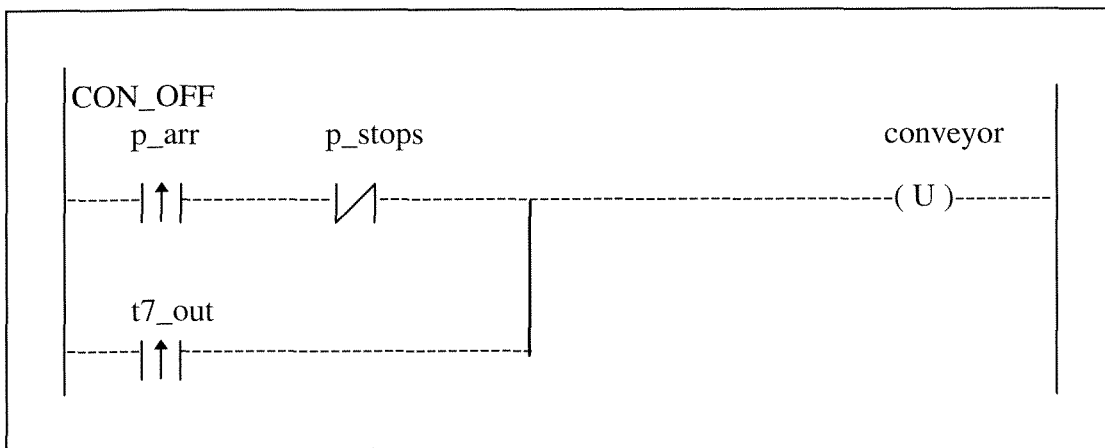


Figure 21 Rung 7

Rung 8 (CPL statement : 7 PalletArrived.WaitOn;
 8 Delay.1000;
 9 PalletLiftUp.Strobe;
 10 PalletLifted.WaitOn;)

This timer rung delays 1000 milliseconds and then lifts the pallet up. This timer runs when the pallet arrives at the loading position. The timer is reset when the pallet is lifted. (Figure 22)

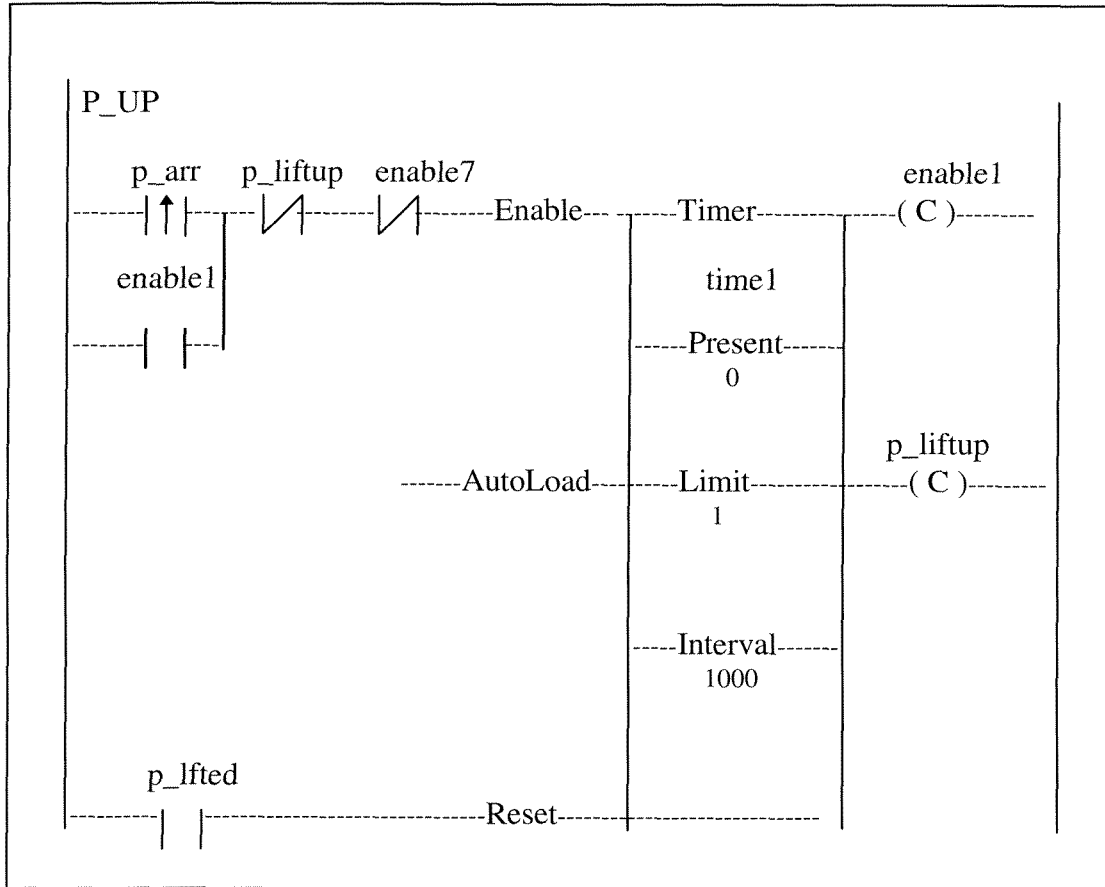


Figure 22 Rung 8

Rung 9 (CPL statement : 9 PalletLiftUp.Strobe;
12 ChuckOpen.Strobe;)

This matrix rung opens the lathe's chuck after the pallet is lifted up. (Figure 23)

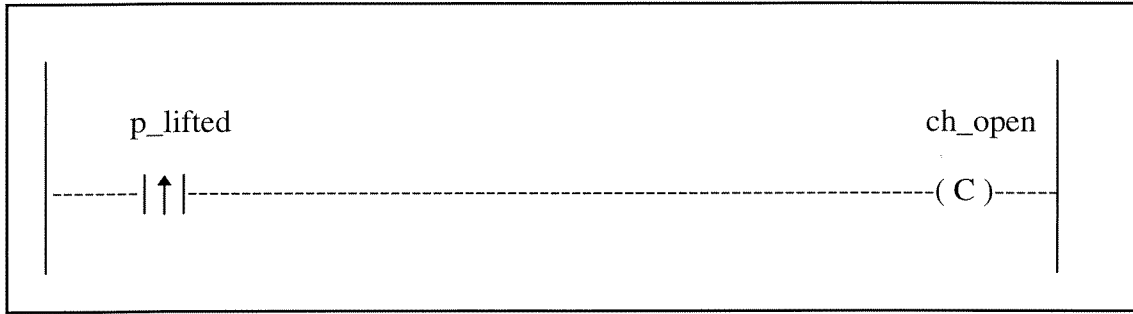


Figure 23 Rung 9

Rung 10 (CPL statement : 9 PalletLiftUp.Strobe;
13 Robot.Do(LoadPart);)

This queue rung starts the process called robot which sends the contents of the file loadpart.cmd to the robot connected to the serial port (Figure 24). This causes the robot to pick up the work piece and move it to the lathe's chuck.

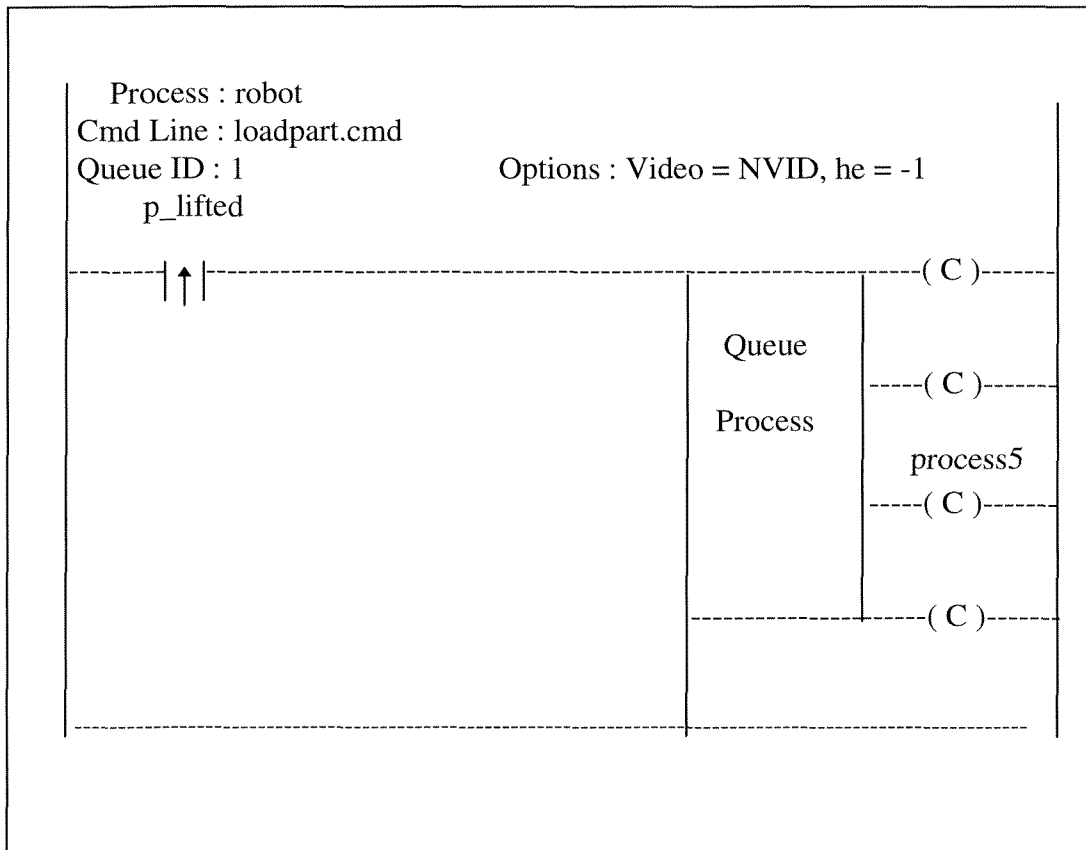


Figure 24 Rung 10

Rung 11 (CPL statement : 14 Delay.1000;
 15 ChuckClose.Strobe;)

This timer rung delays 1000 milliseconds and then closes the chuck on the lathe.
 Process5 is output from the rung 10 when the robot process completes. (Figure 25)

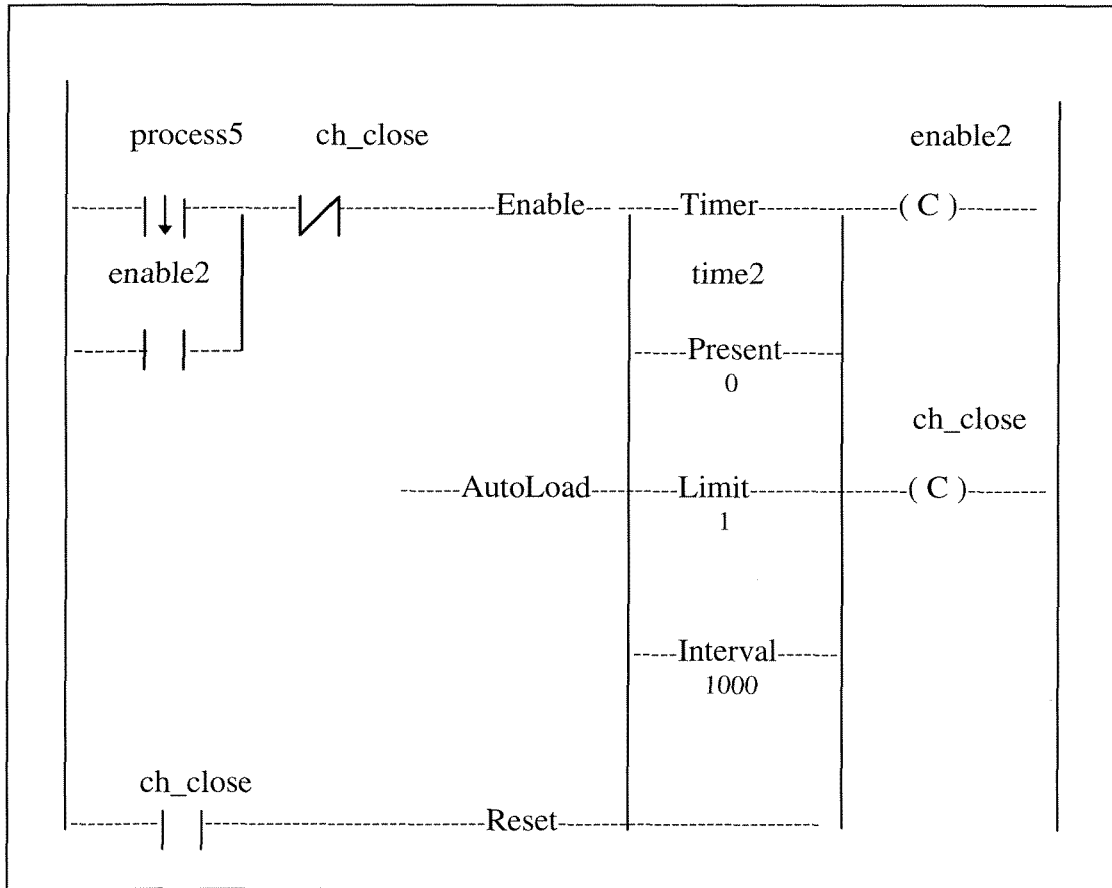


Figure 25 Rung 11

Rung 12 (CPL statement : 15 ChuckClose.Strobe;
16 Delay.2000;)

This timer rung delays 2000 milliseconds after the chuck is closed and before running the next robot process. (Figure 26)

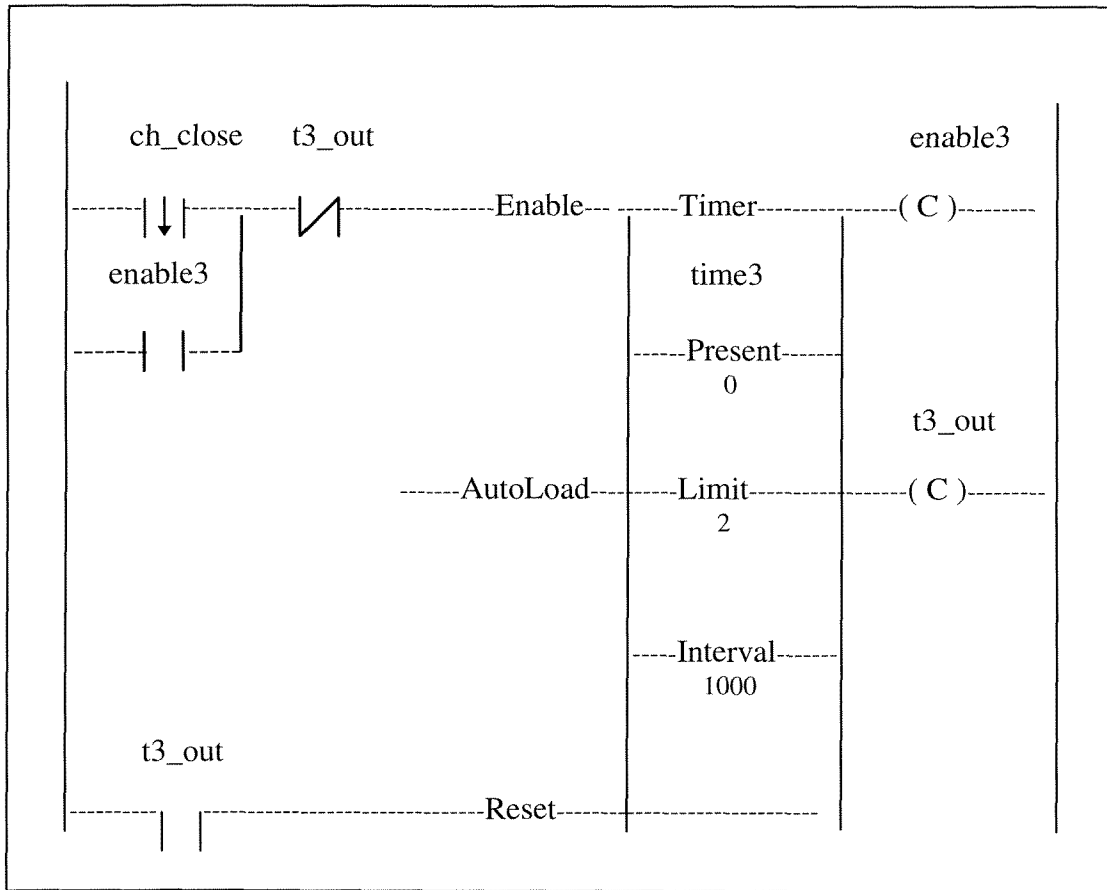


Figure 26 Rung 12

Rung 13 (CPL statement : 16 Delay.2000;
17 Robot.Do(MoveAway);)

This queue rung starts the process called robot to move the robot away from the lathe. It sends the contents of the file `moveaway.cmd` to the robot connected to the serial port. (Figure 27)

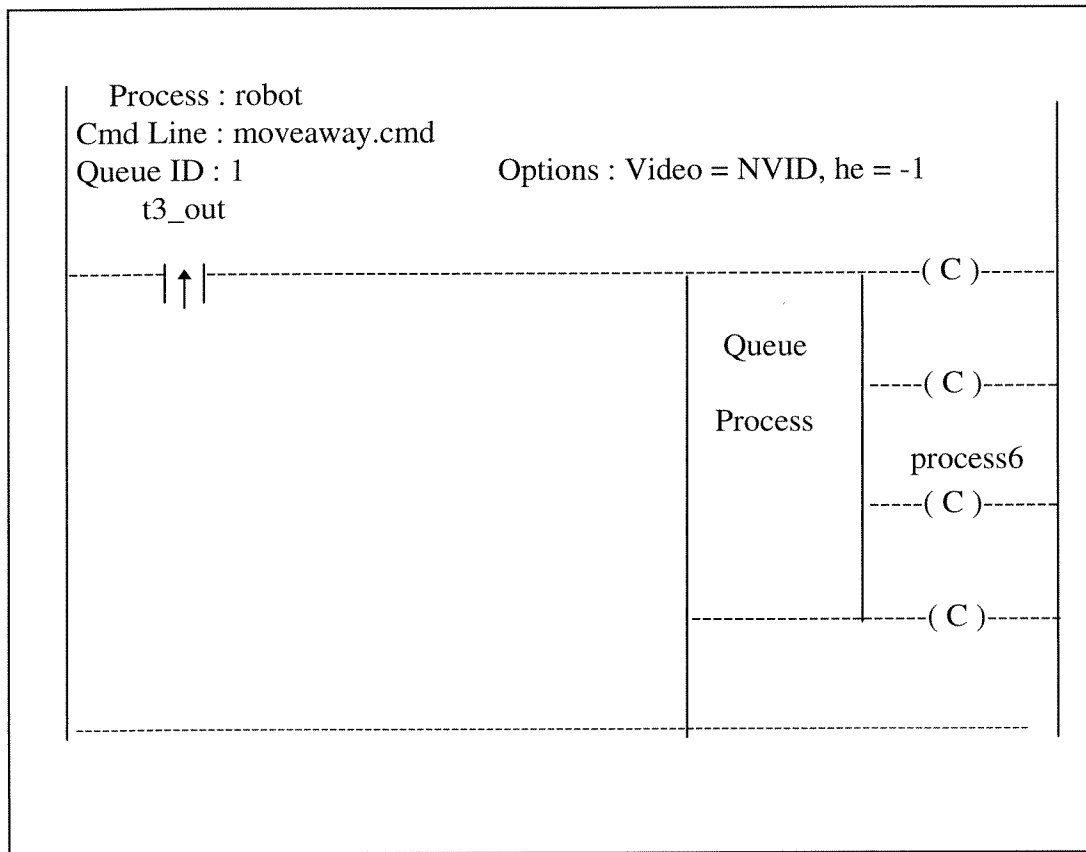


Figure 27 Rung 13

Rung 14 (CPL statement : 18 Delay.2000;
19 LatheStart.Strobe;)

This timer rung delays 2000 milliseconds after moving robot away from the lathe and then starts the lathe. Process6 is output from the rung 13 when the robot process completes. (Figure 28)

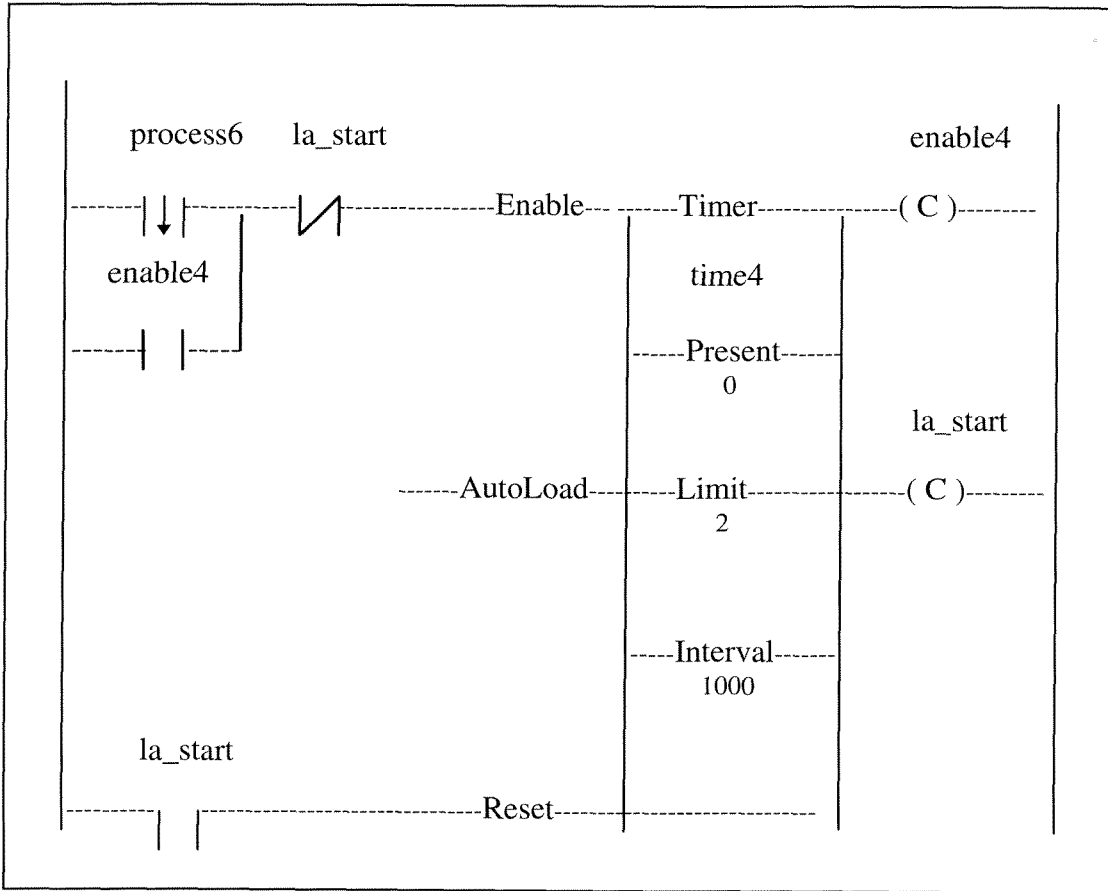


Figure 28 Rung 14

Rung 15 (CPL statement : 20 LatheStop.WaitOff;
21 Robot.Do(MoveBack);)

This queue rung starts the process called robot to move the robot back to the lathe. It sends the contents of the file moveback.cmd to the robot connected to the serial port. (Figure 29)

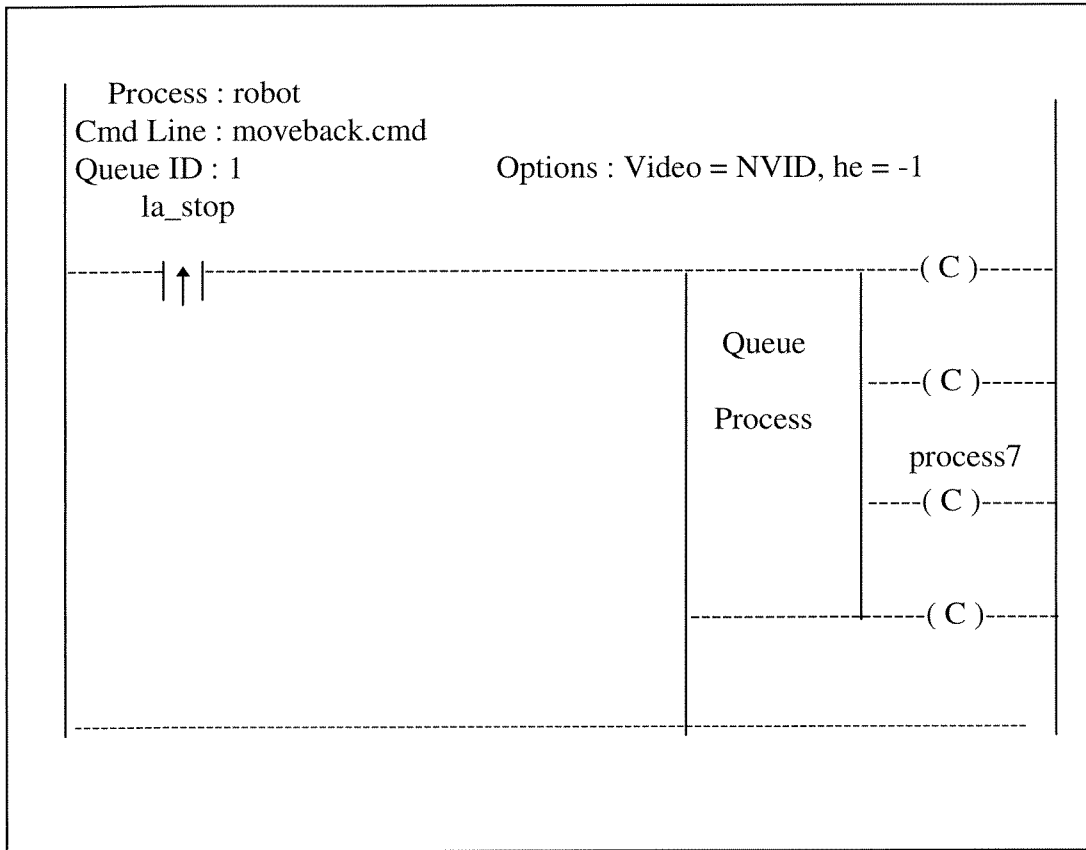


Figure 29 Rung 15

Rung 16 (CPL statement : 22 Delay.2000;
23 ChuckOpen.Strobe;)

This timer rung delays 2000 milliseconds and then opens chuck. Process7 is output from the rung 15 when the robot process completes. (Figure 30)

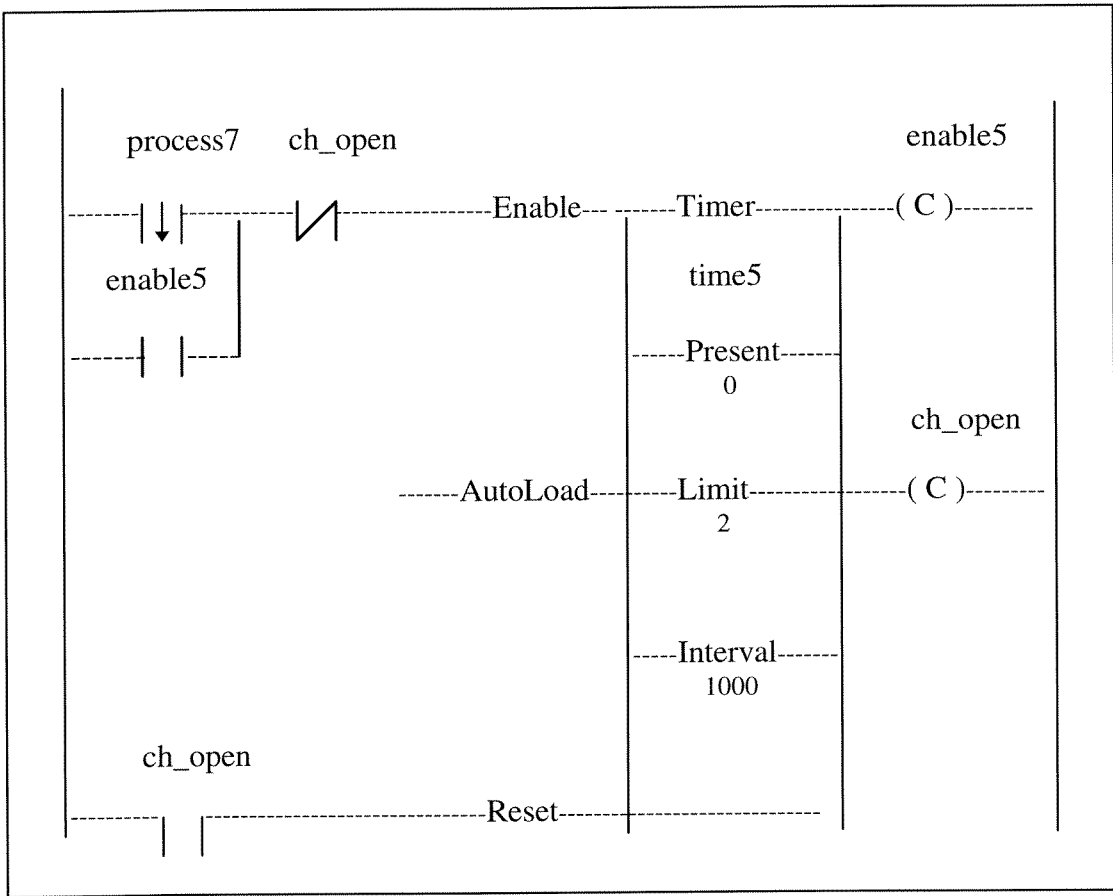


Figure 30 Rung 16

Rung 17 (CPL statement : 23 ChuckOpen.Strobe;
24 Delay.2000;)

This timer rung delays 2000 milliseconds before running the robot's next process.
(Figure 31)

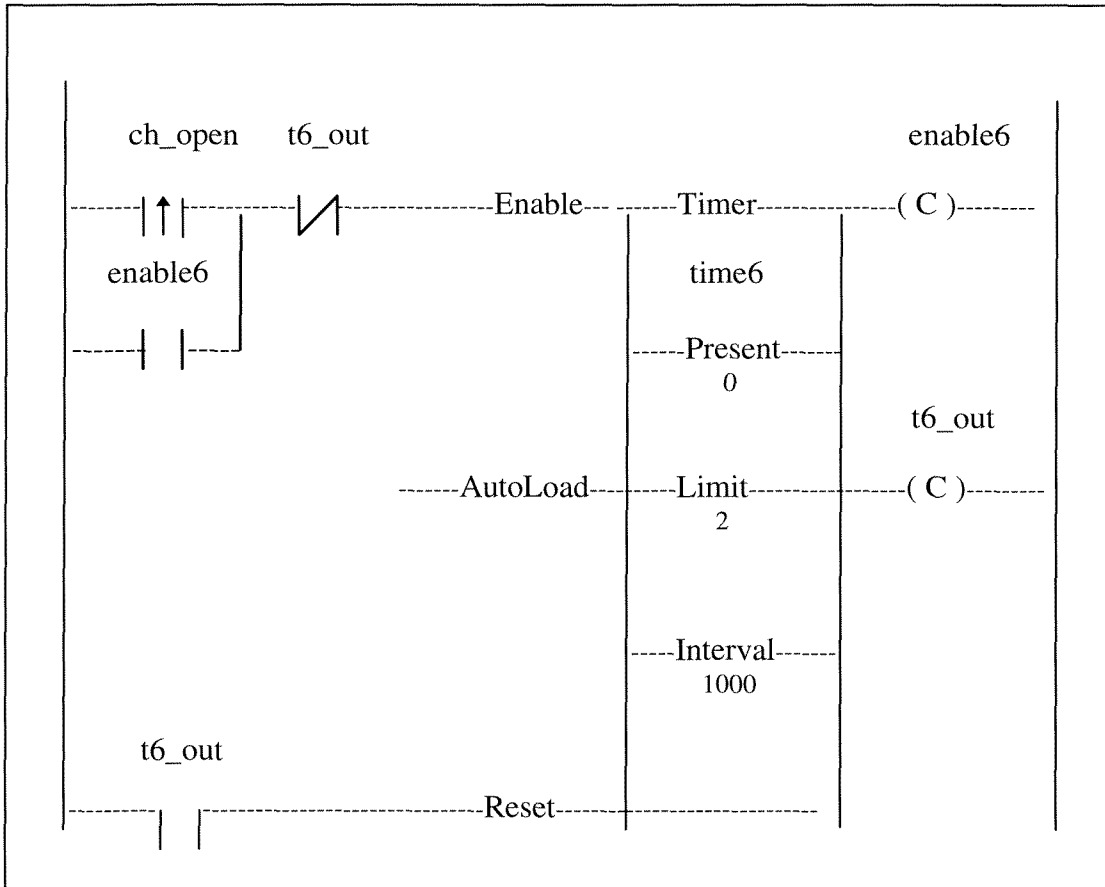


Figure 31 Rung 17

Rung 18 (CPL statement : 25 Robot.Do(GetPart);)

This queue rung starts the process called robot to cause the robot to unload the part by sending the contents of the file gtepart.cmd to the robot connected to the serial port. (Figure 32)

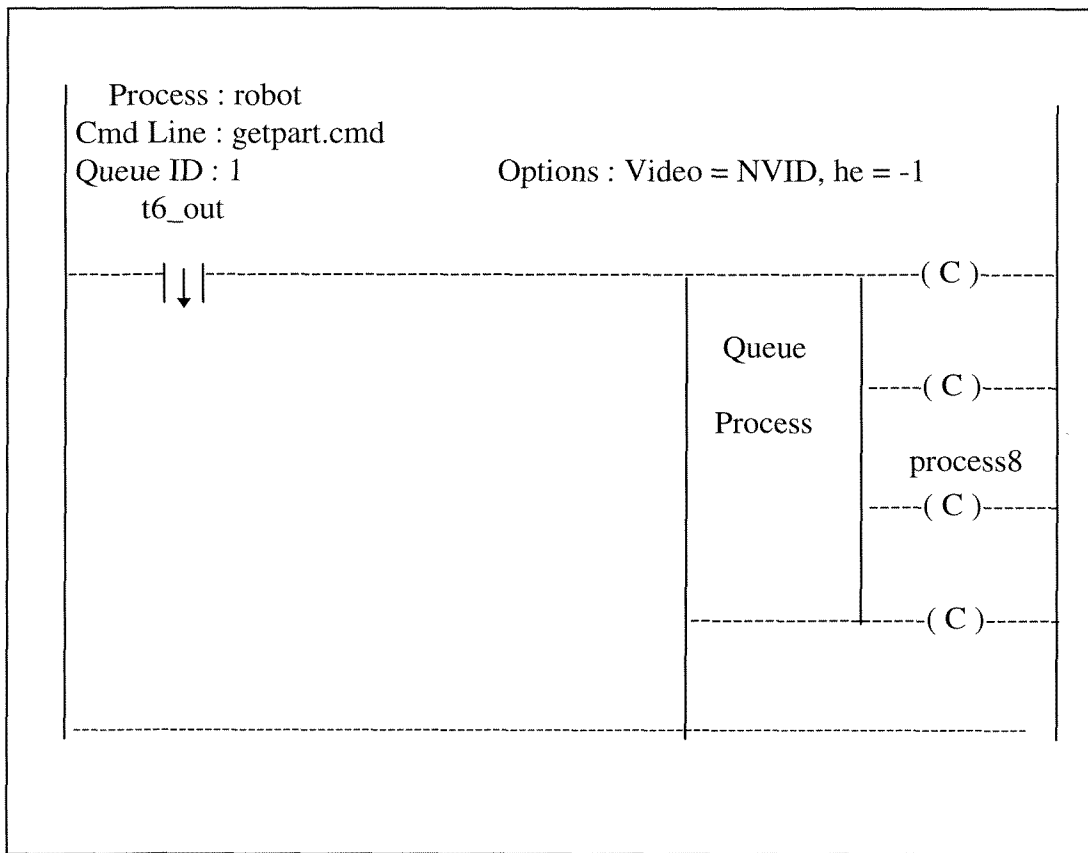


Figure 32 Rung 18

Rung 19 (CPL statement : 27 PalletLiftDown.Strobe;)

This matrix rung moves the pallet lift down. Process8 is output from the rung 18 when the robot process completes. (Figure 33)

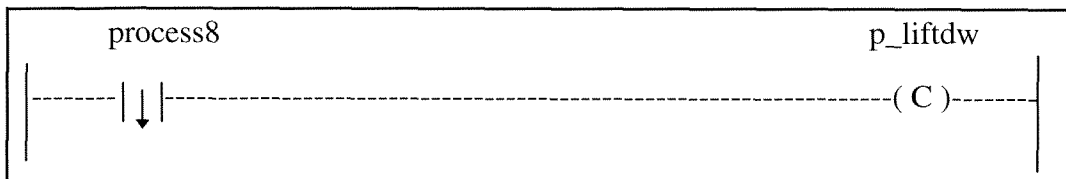


Figure 33 Rung 19

Rung 20 (CPL statement : 27 PalletLiftDown.Strobe;
29 Delay.2000;)

This timer rung delays 2000 milliseconds then shuts down work cell. (Figure 34)

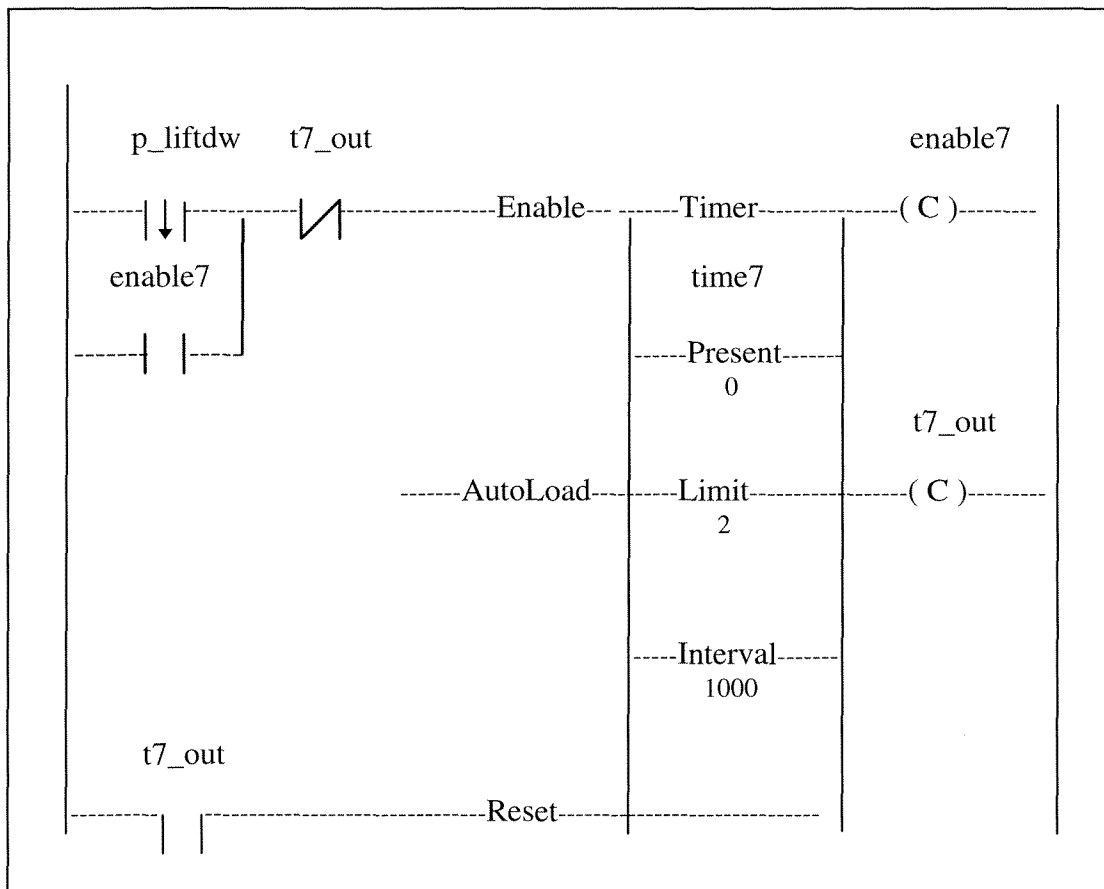


Figure 34 Rung 20

5.3 Interfacing Processes

Figure 14 showed the major components of the OC application developed for this study to be the ladder program and three processes for interfacing with the I/O devices in the cell. Currently the Automated Control System at Manufacturing Engineering Lab consists of two Robots, one CNC Machine, Conveyor, and an Automated Storage and Retrieval System. Some of these devices use discrete I/O signals for interfacing, and the others have to use serial or parallel communications for interfacing. The ladder logic system supported by Omega Controlware cannot directly access the discrete I/O interface or the communications ports. However, Omega supplies some function libraries for various high-level languages, which provide some I/O functions and replace certain DOS capabilities. External processes must then be written, using those functions to do the required I/O. These are described below.

The Omega Controlware functions let a process access the system's hardware (i.e. serial ports) and software resources (i.e. the data table). The system functions that are provided with OC facilitate:

- Process and Thread Control
- Data Item Access and Interprocess Communications
- Even Handling and Timing
- Console I/O
- Printer Control
- Serial Communication
- File Handling
- *Error Handling*

Omega Controlware provides functions and 'include' files for each of the supported high-level languages and assembler language as well. The user can choose the appropriate language to access the OC function. The following are some languages supported by OC:

- IBM Basic Compiler
- Microsoft QuickBASIC Compiler
- Microsoft C Compiler
- Borland C Compiler
- Borland C++ Compiler
- Borland Turbo Pascal
- 8086 Assembler

In the following section the processes shown in Figure 14 are described.

5.3.1 Polling the Acquisition Board

There are discrete I/O signals in our control system, such as, conveyor on/off, photo cell, lathe on/off, pallet up/down, chuck open/close, and some position switches. A 168-Channel Universal Digital I/O Interface is used to send and receive all discrete I/O signals [OmegaDACS-168]. As was mentioned above, it is necessary to design the

interface between OC and the discrete I/O signals so OC can access these I/O signals. This is the purpose of the poll process.

The Omega functions used in the poll process to poll and update the data acquisition board are the following:

SF_RDATA : Read Data Table Item into Local Buffer.

SF_RDATA reads the contents of a specified data table item and places the data into a specified output buffer.

Parameter	Type	Description
item_str	Input String	Name or handle of data table item
size	Input uword	Output buffer size(bytes)
count	Output uword	Number of bytes read
buffer	Input buffer	Point to read data buffer

Table 7. Summary of SF_RDATA

The syntax of SF_RDATA :

```
sf_rdata(item_str, sizeof(size), &count, &buffer);
```

SF_WDATA : Write Local Buffer to Data Table Item.

SF_WDATA copies the contents of a local buffer to a data table item.

Parameter	Type	Description
item_str	Input string	Name or handle of data table item
count	Output uword	Number of bytes to write
buffer	Input buffer	Point of data to be written

Table 8. Summary of SF_WDATA

The syntax of SF_WDATA :

```
sf_wdata(item_str, sizeof(count), &buffer);
```

Figure 35 is the implementation for polling data acquisition board using Borland C++.

```

#include "stdio.h"
#include "dos.h"
#include <stdlib.h>
#include "sf.h"

#define BASE 640

unsigned char data;

void initialize_data_acquisition_board(void)
{
    outportb(BASE+3,144);
    outportb(BASE+1,255);
    outportb(BASE+2,255);
}

void poll(void)
{
    UBYTE port_a;
    UBYTE port_b;
    UBYTE port_c;
    UWORD count;

    // system function, reads the contents of a specified data table item and places the
    // data into a specified output buffer

    sf_rdata("port_b", sizeof(port_b), &count, &port_b);
    sf_rdata("port_c", sizeof(port_c), &count, &port_c);

    data = ~ port_b;
    outportb(BASE+1, data);
    data = ~ port_c;
    outportb(BASE+2, data);
    data = inportb(BASE);
    port_a = ~ data;

    // system function, copies the contents of a local buffer to a data table item
    sf_wdata("port_a", sizeof(port_a), &port_a);
}

main( )
{
    initialize_data_acquisition_board();
    poll();
    return 0;
}

```

Figure 35 The implementation for polling data acquisition board

5.3.2 Communication with Robot and Lathe

The two kinds of programmable devices in the control system are the robot and CNC lathe which are controlled by sending programs to an appropriate interface (either a COM or LPT port). As was mentioned before, the robot and CNC lathe have to be programmed in their host languages. It should be noted that these commands can be separately generated by other software, i.e., CNC commands generated by CAD/CAM software, and then stored in files. This is explained more fully in [Liwu94]. Thus, the end-user can use these files without having to deal with the low-level instructions directly.

The processes that communicate with the robot and lathe to access a special file written in host language commands and send those commands to either the robot or lathe by either serial or parallel port. The process need to get the file name from the ladder program. The Omega function SF-PAR is then used in these process to get the command line arguments, and is described below.

SF_PAR : Get Command Line Parameter.

Parameter	Type	Description
parameter	Input ubyte	Parameter number (1 relative, or 0 for process filespec string)
command_sel	Input ubyte	Command line selector, where: CL_CUR = Current process command line CL_SYS = System process command line
size	Input uword	Size of parameter string buffer (bytes)
length	Output uword	Length of parameter string (bytes)
buffer	Input buffer	Pointer to parameter string buffer (should be 128 bytes)

Table 9. Summary of SF_PAR

The syntax of SF_PAR :

```
sf_far(parameter, command_sel, sizeof(buffer), &length, buffer);
```

Another Omega function used is SF_WAITT which cause a delay.

SF_WAITT : Wait Time Interval.

SF_WAITT causes the thread to be suspended until the specified time interval has elapsed.

Parameter	Type	Description
interval	Input udword	Time interval (milliseconds)

Table 10. Summary of SF_WAITT

The syntax of SF_WAITT :

```
sf_waitt(interval);
```

5.3.3 Robot Software Interface

The robot used in the CIM Lab is a MICRO ROBOT RM-501 (Mitsubishi Electric Corporation) with five axis mobility. RM-501 supports several control modes such as test mode, personal computer mode, edit mode and ROM mode. In personal computer mode the robot can be controlled by the user via personal computer interfaces such as parallel or serial interfaces.

All operations of the RM-501 are controlled by commands. The commands are in a robot language similar to assembly language and make it possible for the users to execute commands concerning robot operation and commands such as condition evaluation. Currently, users program the various robot operations in robot language off-line, save the commands in a file, and later, when needed to operate the robot, download the file into the robot through either a parallel or serial port and then execute the commands. [RM501, Liwu94].

Figure 36 is the process developed for communicating with the robot using Borland C++ via the serial port.

```

#include <stdio.h>
#include <bios.h>
#include <conio.h>
#include "sf.h"

#define SIZE 256
#define COM2 1

void initialize_serial_port(void)
{
    int com_port, baud_rate, character_bit, parity, stop_bit;

    com_port = COM2;
    baud_rate = _COM_300;
    character_bit = _COM_CHR8;
    parity = _COM_NOPARITY;
    stop_bit = _COM_STOP1;

    _bios_serialcom(_COM_INIT, com_port, baud_rate | character_bit | stop_bit | parity);
}

main( )
{
    FILE *fp;
    char *ip, buffer[SIZE];
    int i;
    int len;
    unsigned status;

    UBYTE parameter;
    UBYTE command_sel;
    UWORD size;
    UWORD length;
    char buffer1[80];
    initialize_serial_port();

    //Get Command Line Parameter from the Cmd Line field of the queue rung.
    //The Cmd Line field specifies a command line string to be passed to the process.
    //This command line can be accessed by the process in the same manner that a
    //program started under DOS can access a command line, in C the argc and argv
    //variables can be used. In all languages, the SF_PAR(get command line
    //parameter) system functions can be used to access the command line.
    parameter = 1;
    command_sel = CL_CUR;
    sf_par(parameter, command_sel, sizeof(buffer1), &length, buffer1);

    // Open the specified file
    if ((fp = fopen(buffer1, "r" )) == NULL )
    {
        printf("\7file %s cannot be opened\n", buffer1);
        exit();
    }

    // Process the contents of the specified file and transfer them to RS-232C port
    fgets(buffer, SIZE, fp);
}

```

```

while (!feof(fp))
{
    /* skip leading blanks */
    for(ip = buffer; *ip == ' ' && *ip != '\0'; ip++);
    strcpy(buffer, ip);
    len = strlen(buffer);
    buffer[len-1] = '\r';
    buffer[len] = '\n';
    buffer[len+1] = '\0';

    for(i = 0; i < len+1; i++)
    {
        for( ; )
        {
            status = _bios_serialcom(_COM_STATUS, COM2, 0);
            if(status & 0x2000)
            {
                _bios_serialcom(_COM_SEND, COM2, buffer[i]);
                break;
            }
        }
        fgets(buffer, SIZE, fp);
    }
    fclose(fp);
    return 0;
}

```

Figure 36 The implementation for communicating with robot

5.3.4 CNC Machine Software Interface

The CNC machine we used at CIM Lab is EMCO COMPACT 5 CNC Lathe. The CNC Lathe comes with two accessories : DNC-Interface and RS-232C-Interface.

The DNC-Interface is a proprietary interface consisting of some special instructions and status, by which the CNC could be controlled by an external computer . The DNC-Interface specification is shown below in the Table 11.

X62/PIN	1	A	Status hand
	2	E	Turret - hand operation
	3	E	Instruction G66 + INP
	4	-	-
	5	-	-
	6	E	Instruction G66 + FWD
	7	A	Status program running
	8	A	Status intermediate stop
	9	E	Instruction switch hand / CNC
	10	-	-
	11	-	-
	12	-	-
	13	-	-
	14	-	-
	15	A	Output set with M8, M9
	16	-	-
	17	E	Instruction start
	18	A	Output set with M22, M23
	19	A	Status main motor ON/OFF
	20	A	Output impulse set with M26
	21	E	Instruction blockage-turret
	22	V	+10V not controlled
	23	V	GND
	24	V	GND
	25	V	GND
	26	V	+5V controlled

Table 11. DNC-Interface specification

In this project, the DNC-Interface is used to control the CNC Lathe working from the independent computer instead of at the CNC 5. Currently, the PIN3 (Instruction G66 + INP) and PIN17 (Instruction start) were used in our control system. The *instruction G66 + INP* (PIN3) was used to set the RS232C mode so that the lathe program can be downloaded into the lathe, and the *instruction start* is used to start the lathe working.

Figure 37 is the process used to communicate with the CNC machine using the serial interface.


```

#include <stdio.h>
#include <bios.h>
#include <conio.h>
#include "sf.h"

#define SIZE 256
#define BASE 640
#define COM1 0

void initialize_serial_port(void)
{
    int com_port, baud_rate, character_bit, parity, stop_bit;

    com_port = COM1;
    baud_rate = _COM_300;
    character_bit = _COM_CHR8;
    parity = _COM_NOPARITY;
    stop_bit = _COM_STOP1;

    _bios_serialcom(_COM_INIT, com_port, baud_rate | character_bit | stop_bit | parity);
}

void set_lathe_mode(void)
// Set the lathe working mode as RS-232C via DNC-Interface by sending instruction G66 + INP
{
    unsigned char data, bit, mask;
    /* lathe handshake */
    printf("Do lathe handshake\n");
    // Set port B, bit 0
    data = inportb(BASE+1);
    data = ~ data;
    bit = 0;
    mask = 1 << bit;
    data = data | mask;
    data = ~ data;
    outportb(BASE+1, data);

    // Causes the thread to be suspended until the specified time interval has elapsed
    // Under the OC, some certain DOS capabilities such as delay function in C have
    // to replace by OC system function.
    sf_waitt(250L);          /* delay for 250 milliseconds */

    // Reset port B, bit 0
    data = inportb(BASE+1);
    data = ~ data;
    bit = 0;
    mask = 1 << bit;
    mask = ~ mask;
    data = data & mask;
    data = ~ data;
    outportb(BASE+1, data);

    /* lathe G66inp */
    sf_waitt(100L);        /* delay 100 milliseconds */
    printf("Do lathe G66inp\n");
}

```

```

// Set port b, bit 1
data = inportb(BASE+1);
data = ~ data;
bit = 1;
mask = 1 << bit;
data = data \ mask;
data = ~ data;
outportb(BASE+1, data);

sf_waitt(800L);          /* delay 800 milliseconds */

// Reset port B, bit 1 */
data = inportb(BASE+1);
data = ~ data;
bit = 1;
mask = 1 << bit;
mask = ~ mask;
data = data & mask;
data = ~ data;
outportb(BASE+1, data);
}

main( )
{
    FILE *fp;
    char *ip, buffer[SIZE];
    int len;
    int i;
    int len;
    unsigned status;

    UBYTE parameter;
    UBYTE command_sel;
    UWORD size;
    UWORD length;
    char buffer1[80];

    initialize_serial_port();
    set_lathe_mode();

    // Get Command Line Parameter from the Cmd Line field of the queue rung.
    // The Cmd Line field specifies a command line string to be passed to the process.
    // This command line can be accessed by the process in the same manner that a
    // program started under DOS can access a command line, in C the argc and argv
    // variables can be used. In all languages, the SF_PAR(get command line
    // parameter) system functions can be used to access the command line.
    parameter = 1;
    command_sel = CL_CUR;
    sf_par(parameter, command_sel, sizeof(buffer1), &length, buffer1);

    // Open the specified file
    if ((fp = fopen(buffer1, "r" )) == NULL )
    {
        printf("\7file %s cannot be opened\n", buffer1);
        exit();
    }
}

```

```

}

// Process the contents of the specified file and transfer them to RS-232C port
fgets(buffer, SIZE, fp);
while (!feof(fp))
{
    len = strlen(buffer);

    for(i = len-1; i<30; i++)
        buffer[i] = ' ';

    buffer[30] = '\r';
    buffer[31] = '\n';
    buffer[32] = '\0';

    for(i = 0; i < 32; i++)
    {
        for( ; ;)
        {
            status = _bios_serialcom(_COM_STATUS, COM1, 0);
            if (status & 0x2000)
            {
                _bios_serialcom(_COM_SEND, COM1, buffer[i]);
                break;
            }
        }
    }
    fgets(buffer, SIZE, fp);
}
fclose(fp);
return 0;
}

```

Figure 37 The implementation for communicating with CNC machine

5.3.5 Communication with Automated Storage and Retrieval System

The Amatrol Automated storage and Retrieval system (AS/RS) is designed to feed automated manufacturing systems. The 862-AS/RS is capable of operating in two modes; one using discrete I/O signals for interfacing, and the other using communications for interfacing [AMATROL91].

The communications mode of the 862-AS/RS is for applications in computer integrated manufacturing (CIM) where real time storage and retrieval is needed. This mode allows an external controller to request the storage or retrieval of a specific type of part.

The storage/retrieval commands are transmitted to and from the 862-AS/RS using 2400 BAUD, 8 data bits, 1 stop bit and no parity. The communication port is a standard RS-232 serial port using pins 2 (transmit), 3 (receive) and 7 (ground). The port does not use hardware handshaking.

The software interface needed to communicate to AS/RS through Omega Controlware is very similar to communication with the robot and CNC machine. A possible process is shown in the Figure 38.

```

#include <stdio.h>
#include <bios.h>
#include <conio.h>
#include "sf.h"

#define SIZE 256
#define COM2 1

void initialize_serial_port(void)
{
    int com_port, baud_rate, character_bit, parity, stop_bit;

    com_port = COM2;
    baud_rate = _COM_2400;
    character_bit = _COM_CHR8;
    parity = _COM_NOPARITY;
    stop_bit = _COM_STOP1;

    _bios_serialcom(_COM_INIT, com_port, baud_rate | character_bit | stop_bit | parity);
}

main( )
{
    FILE *fp;
    char *ip, buffer[SIZE];
    int i;
    int len;
    unsigned status;

    UBYTE parameter;
    UBYTE command_sel;
    UWORD size;
    UWORD length;
    char buffer1[80];
    initialize_serial_port();

    //Get Command Line Parameter from the Cmd Line field of the queue rung.
    //The Cmd Line field specifies a command line string to be passed to the process.
    //This command line can be accessed by the process in the same manner that a
    //program started under DOS can access a command line, in C the argc and argv
    //variables can be used. In all languages, the SF_PAR(get command line
    //parameter) system functions can be used to access the command line.
    parameter = 1;
    command_sel = CL_CUR;
    sf_par(parameter, command_sel, sizeof(buffer1), &length, buffer1);

    // Open the specified file
    if ((fp = fopen(buffer1, "r" )) == NULL )
    {
        printf("\7file %s cannot be opened\n", buffer1);
        exit();
    }

    // Process the contents of the specified file and transfer them to RS-232C port
    fgets(buffer, SIZE, fp);
}

```

```

while (!feof(fp))
{
    len = strlen(buffer);
    for(i = 0; i < len+1; i++)
    {
        for(;;)
        {
            status = _bios_serialcom(_COM_STATUS, COM2, 0);
            if (status & 0x2000)
            {
                _bios_serialcom(_COM_SEND, COM2, buffer[i]);
                break;
            }
        }
    }
    fgets(buffer, SIZE, fp);
}
fclose(fp);
return 0;
}

```

Figure 38 The Possible Interface Implementation for the AS/RS

6. Comparison of Ladder Logic language vs CPL

6.1 Strengths of RLL over CPL

The RLL is a graphic language, based on one of the programming standards (Language LD). There are a number of professional technicians dealing with this language and there are many RLL software development tools available today, like Omega Controlware.

The versatility of the RLL enables it to be used in the control of a wide range of machines and various applications such as oil refinery control, traffic control, machine control, effluent flow, packaging line control, etc. RLL-based programmable controllers have been replacing old-style relay logic with increasing rapidity for more than a decade until today they have rendered relay logic obsolete. It is the most widely used and accepted industrial control language [Pessen89].

There are some unique strengths for Omega Controlware. Basic to the design and use of Omega Controlware are multitasking and shared data. The multitasking capabilities of OC are unique because the system was specifically designed to support the needs of control applications rather than the general purpose needs of data processing. The data sharing capabilities of Omega Controlware are very similar to the capabilities of many conventional multitasking systems but with some extensions dictated by the inclusion of networking and global I/O within a control environment. Also, Omega can in real time monitor the status of all work cells in the control system.

CPL lacks the ability to scan more than one input at a time. Its procedure is a strict sequence of operations. For example, while CPL is waiting for an input from a sensor, all other I/Os are ignored. This means that CPL cannot wait for two or more events concurrently and reacts to the one that comes first. This is in contrast to RLL which can concurrently scan multiple inputs and react to them as they occur.

There are some differences in structure between a RLL program and CPL. For example, in Figure 18, one RLL rung combines several CPL steps and it is easy to find all conditions that turns on the pallet_stop. In Figure 9, pallet_stop statements have to spread out in lines 3 and 26 by CPL.

6.2 Weaknesses of RLL using OC

In reality RLL is not standardized. Each PLC manufacturer has implemented RLL in different ways. Thus RLL programs are not portable and programmers must be retrained when using different PLCs.

Large RLL programs become difficult to understand and follow because the RLL language is unstructured and the quality of RLL programming depends on some experience in a control environment.

To use Omega Controlware, besides RLL code, it is necessary to implement some custom software to interface with the data acquisition board, the robot and lathe in a language such as C. Also, some OC system functions replace certain DOS capabilities. In such cases it is necessary to use an OC system function. For example, the Borland C++ compiler has a function called delay that suspends execution for an interval (milliseconds). If this function is needed, it must be replaced with an OC system function called WAITT. It would not be feasible for most FMS programmers to be expected to learn the C language and Omega functions to write such interface programs.

In comparison to RLL, a simple CPL program is much easier to read and modify. There are some weaknesses in RLL structure compared to CPL. For example, in Figure 9 it is easy to see in line 26 all steps (1-25) that have been executed, but in RLL Appendix A this is not obvious. Also, in programming RLL the code needs an extra input condition to control the sequence of operations, whereas in CPL it is implicit that one step follows the next step. For example, in Figure 18, RLL requires one more input condition (start) to control the sequence of the operations that follow.

7. Conclusion

This project demonstrated that RLL can be used to replace CPL as a programming language in the Manufacturing Engineering CIM Lab. The RLL code based on Omega Controlware has been successfully tested for correctness.

We could further develop our control system and add more workcells in FMS such as Amatrol Automated Storage and Retrieval System, Robot Vision System etc. With the feature of multitasking and networking of OC we could in real time load various programs, which are written in the host command languages such as Robot and CNC machine, from some remote machine into the device controller.

Compared to CPL, Omega Controlware has some advantages and disadvantages. With the PRO, a state of the art editor for the development of Omega Controlware RLL programs, it is easily to program RLL code once the user is trained in RLL. Students of the Department of Manufacturing engineering should be familiar with RLL, the most popular control language in manufacturing engineering. However, training in RLL takes more time than learning CPL, since CPL is much easier to read and understand. A major advantage to using RLL is that it supports scanning of many simultaneous inputs, which CPL does not support. Thus, sophisticated control programs can be written in RLL but not in CPL. However, to use Omega Controlware, technical support is required to maintain the external interface processes and to develop new interface processes when these become necessary. This requirement may be a significant obstacle for continued use of Omega. Table 12 summarizes the strengthes and weaknesses of CPL vs RLL.

Characteristic	CPL	Omega RLL
Ease of learning	Easier	
Ease of programming	Easier	
Ease of adding new elements		More capabilities
Ease of modification	Easier to modify	
Technical support	None	Commercial product
Real-time	Slower	Faster execution
Sophisticated system control	Weak	More capabilities
Standard industrial control languages	Not standard	Use of RLL
Multitasking & networking	None	Supports both
Portable	CPL not portable	RLL may be portable
Additional external programming	Not required	Required
PC memory		Requires more RAM
Needed programming skills	CPL only	Knowledge of RLL and C

Table 12 Summarization of CPL and RLL

References

[ALLEN-BRADLEY87] User's manual, Bulletin 1745, SLC Programmable Controllers, Publication 1745-800-November, 1987.

[AMATROL91] AMATROL 862-AS/RS Communication Protocol.

[Chaar90] Chaar, J. K. A Methodology for Developing Real-Time Control Software for Efficient and Dependable Manufacturing Systems, Ph.D Dissertation, University of Michigan, 1990

[Benhabib89] Benhabib, B., Chen, C.Y., Johnson, W.R., An Integrated Manufacturing Work Cell Management System, Manufacturing Review Vol 2, No 4, December 1989.

[EMCO114004] EMCO Compact 5 CNC Software A6C 114004.

[IEC 1131-1] Project CEI 1131-3 : Automates programmables, IEC DIS 1131-3.

[Martin89] Martin, J.M. (1989), Cells drive manufacturing strategy, Manufacturing Engineering, January, 49-54.

[Meghamala92] Development of an Object-Oriented High-Level Language and Construction of an Associated Object-Oriented Compiler, working paper #92-105, System Analysis Department, Miami University, Oxford, Ohio, 1992.

[Meghamala92] Meghamala, N., "Development of an Object-Oriented High-Level Language and Construction of an Associated Object-Oriented Compiler", Technical Report, Systems Analysis Department, Miami University, 1992.

[OmegaDACS-168] Omega, Data Acquisition and Control System, 168-Channel Universal Digital I/O Interface for IBM PC/XT/AT.

[Omega903-19992-1.00R] Omega Controlware User's Guide, Document Number 903-19992-1.00R.

[Omega903-19992-1.09V] Omega Controlware reference Guide, Document Number 903-19992-1.09V.

[Pessen89] D. W. Pessen, Ladder-diagram Design for Programmable Controllers, Automatics, Vol. 25, No. 3, pp. 407-412, 1989.

[RM501] RM501 Manual, Mitsubishi Electric Corporation.

Appendix A Complete RLL Program Using Omega Controware

Use OMEGA Controlware to control the CIM cell

Friday, May 12, 1995 6:06 pm

Mark H. Ma
Systems Analysis
Miami University
Lathe : COM1
Robot : COM2

STATION\\C:\OC\test6.oc
Friday, May 12, 1995 6:06 pm
Use OMEGA Controlware to control the CIM cell

Table of Contents

Application File Listing	1
Root Ladder	1
Items 1-5	1
Items 6-18	2
Items 19-31	3
Items 32-33	4
Items 34-35	5
Items 36-38	6
Items 39-40	7
Items 41-42	8
Items 43-44	9
Items 45-46	10
Items 47-48	11
Items 49-50	12
Items 51-52	13
Ladder Directory	14
Item Directory	15
Name Directory	19

Title 1
Use OMEGA Controlware to control the CIM cell
Mark H. Ma
Systems Analysis
Miami University
Lathe : COM1
Robot : COM2

Data 2
Ubyte port_a (CIM1) ' CIM Cell Control Input Module '
.0
.1
.2
.3
.4 (La_Stop) ' lathe Stop '
.5 (P_Lifted) ' Pallet Lifted '
.6 (P_Arr) ' Pallet Arrived '
.7 (Pho_Cell) ' Photo Cell '

Data 3
Ubyte port_b (CIM2) ' CIM Cell Control Output Module '
.0 (La_hand) ' Lathe Handshank '
.1 (La_G66) ' Lathe G66inp '
.2 (La_Run) ' lathe Running '
.3
.4
.5
.6
.7

Data 4
Ubyte port_c (CIM3) ' CIM Cell Control Output Module '
.0 (P_stops) ' Pallet Stops '
.1 (Ch_open) ' Chuck Open '
.2 (La_start) ' Lathe Start '
.3 (Ch_close) ' Chuck Close '
.4 (P_liftup) ' Pallet Lift Up '
.5 (Conveyor) ' Conveyor '
.6 (P_liftdw) ' Pallet Lift Down '
.7

Data 5
Bit process2 ' Run process robot (nest.cmd) '

Data 6 Bit process3	' Run process lathe1 '
Data 7 Bit process4	' Run process poll '
Data 8 Bit process5	' Run process robot (loadpart.cmd) '
Data 9 Bit process6	' Run process robot (moveaway.cmd) '
Data 10 Bit process7	' Run process robot (moveback.cmd) '
Data 11 Bit process8	' Run process robot (getpart.cmd) '
Data 12 Bit process9	' Run process reset '
Data 13 Bit start	' start conveyor running '
Data 14 Bit enable1	' time1 enable '
Data 15 Bit enable2	' time2 enable '
Data 16 Bit enable3	' time3 enable '
Data 17 Bit enable4	' time4 enable '
Data 18 Bit enable5	' time5 enable '

Data 19
bit enable6 ' time6 enable '

Data 20
bit enable7 ' time7 enable '

Data 21
bit t3_out ' time3 output '

Data 22
bit t6_out ' time6 output '

Data 23
bit t7_out ' time7 output, shut down machine '

Data 24

Data 25
WORD time1 ' Delay 1000 miliseconds then lift Pallet up '

Data 26
WORD time2 ' delay 1000 miliseconds then get chuck close '

Data 27
WORD time3 ' delay 2000 miliseconds '

Data 28
WORD time4 ' delay 2000 miliseconds then start lathe '

Data 29
WORD time5 ' delay 2000 miliseconds then open chuck '

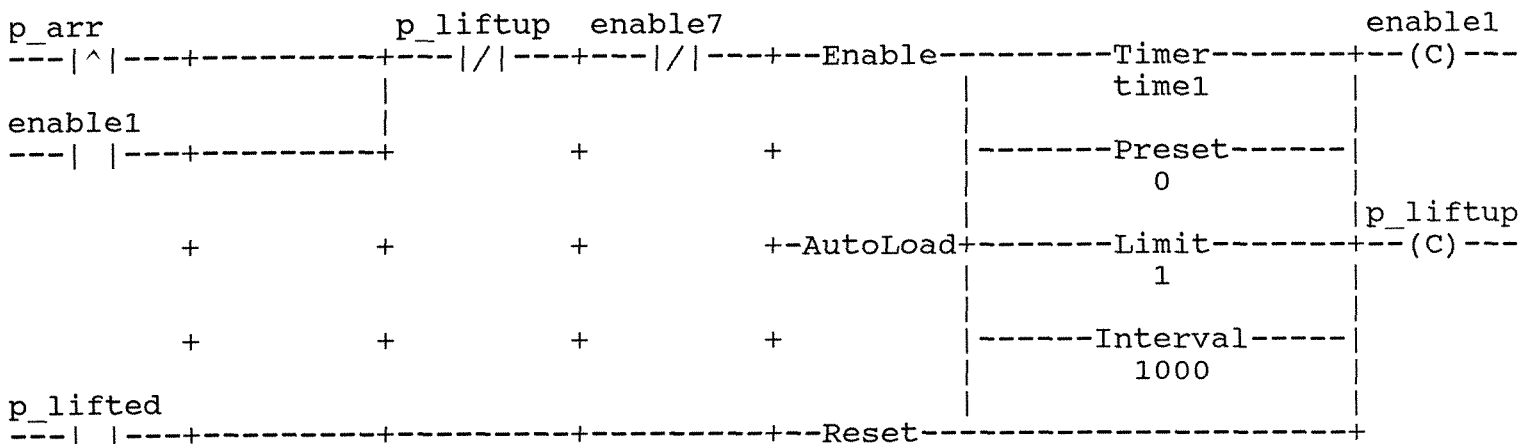
Data 30
WORD time6 ' delay 2000 miliseconds '

Data 31
WORD time7 ' delay 500 miliseconds '

rung 39

This timer rung delays 1000 milliseconds then gets pallet up

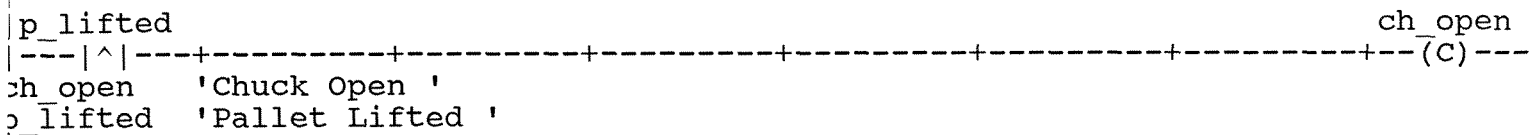
P_UP



enable1 'time1 enable '
 enable7 'time7 enable '
 p_arr 'Pallet Arrived '
 p_lifted 'Pallet Lifted '
 p_liftup 'Pallet Lift Up '
 time1 'Delay 1000 milliseconds then lift Pallet up '

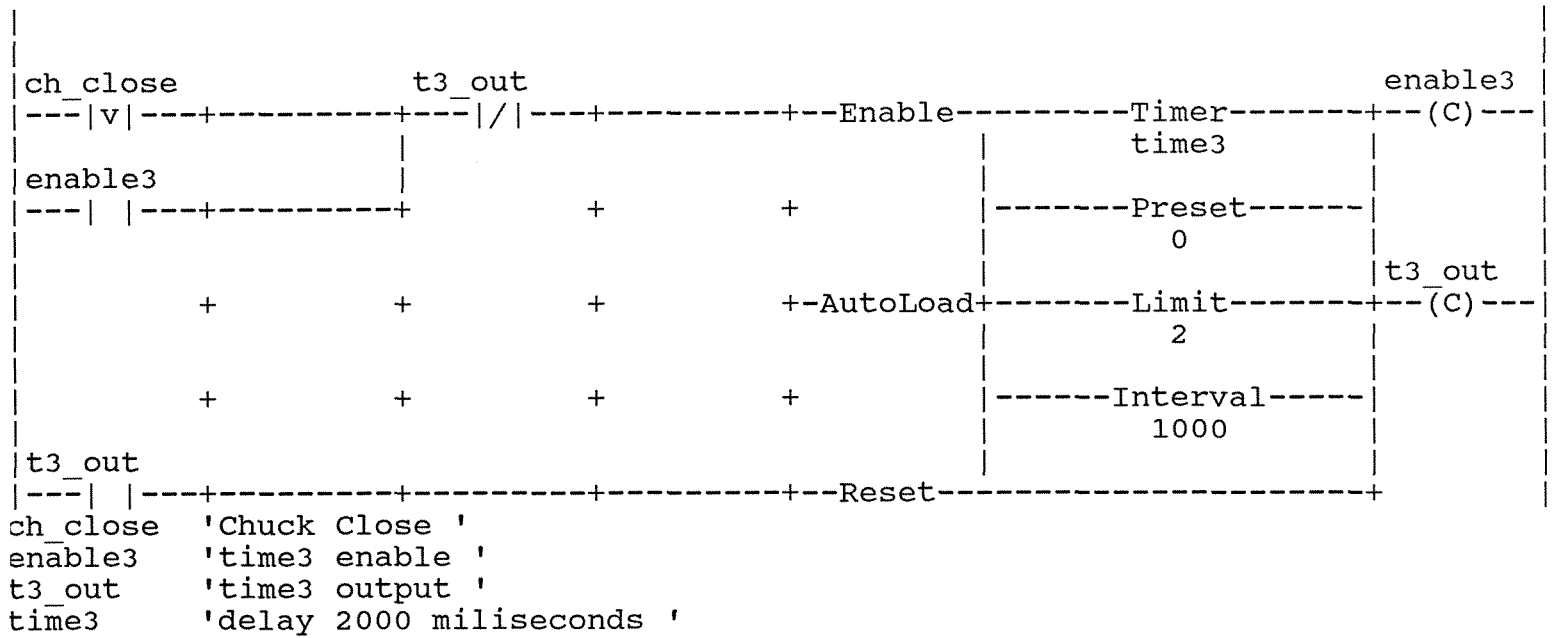
rung 40

This matrix rung open the lathe's chuck



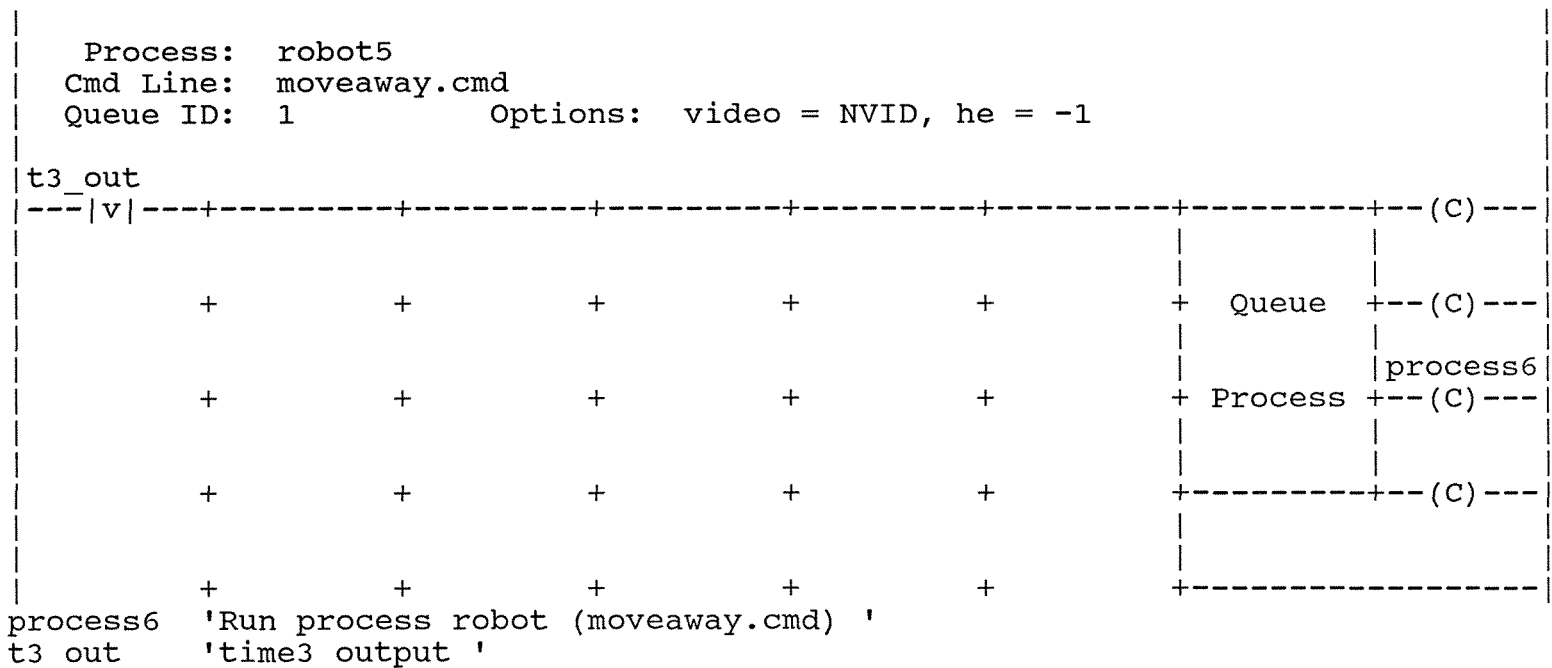
Rung 43

This time rung delay 2000 miliseconds in order to run robot's process



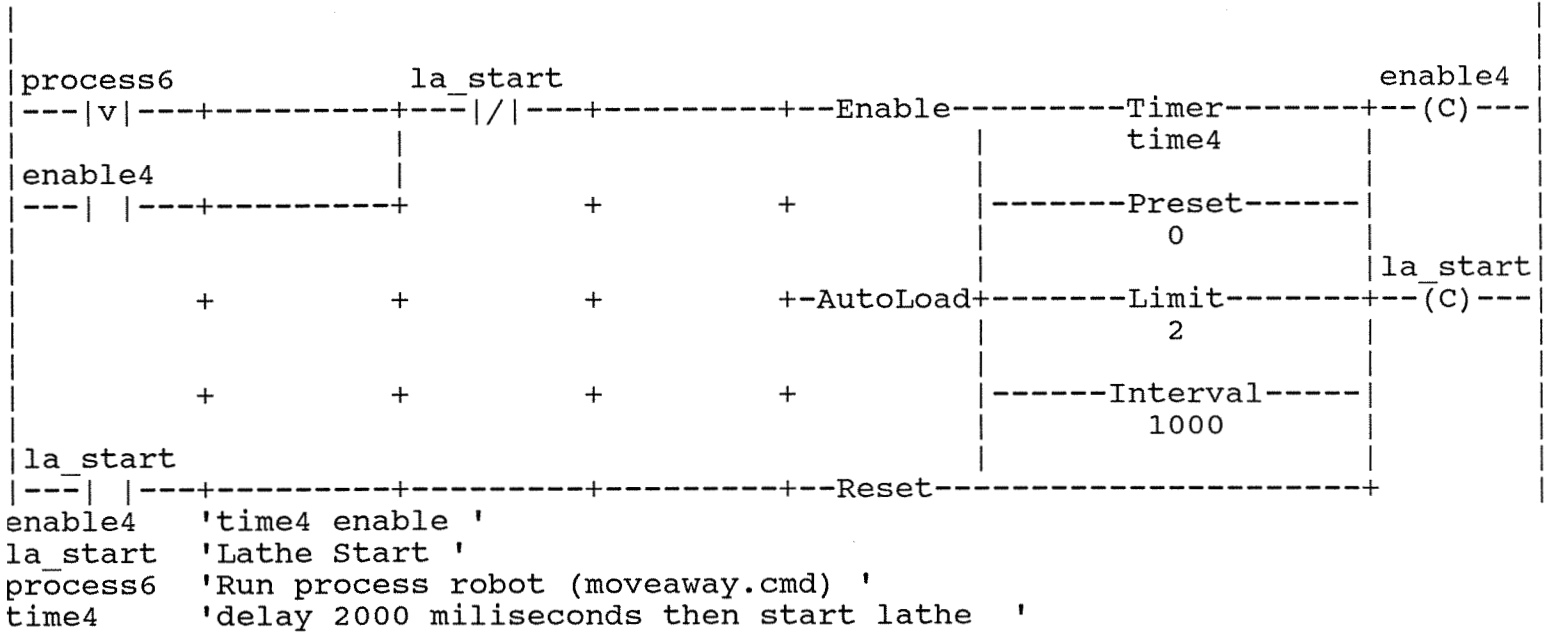
Rung 44

This queue start the process that moves robot away



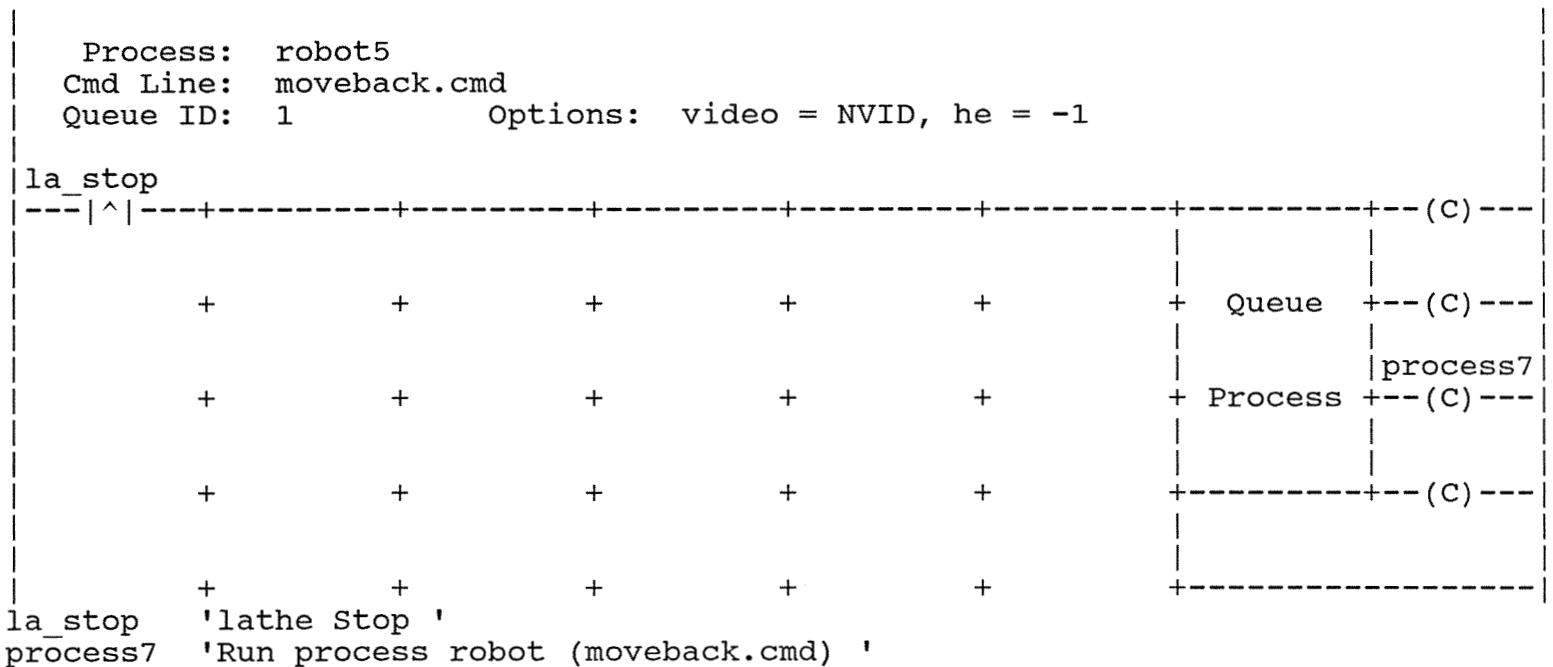
Rung 45

This timer rung delay 2000 miliseconds then start lathe



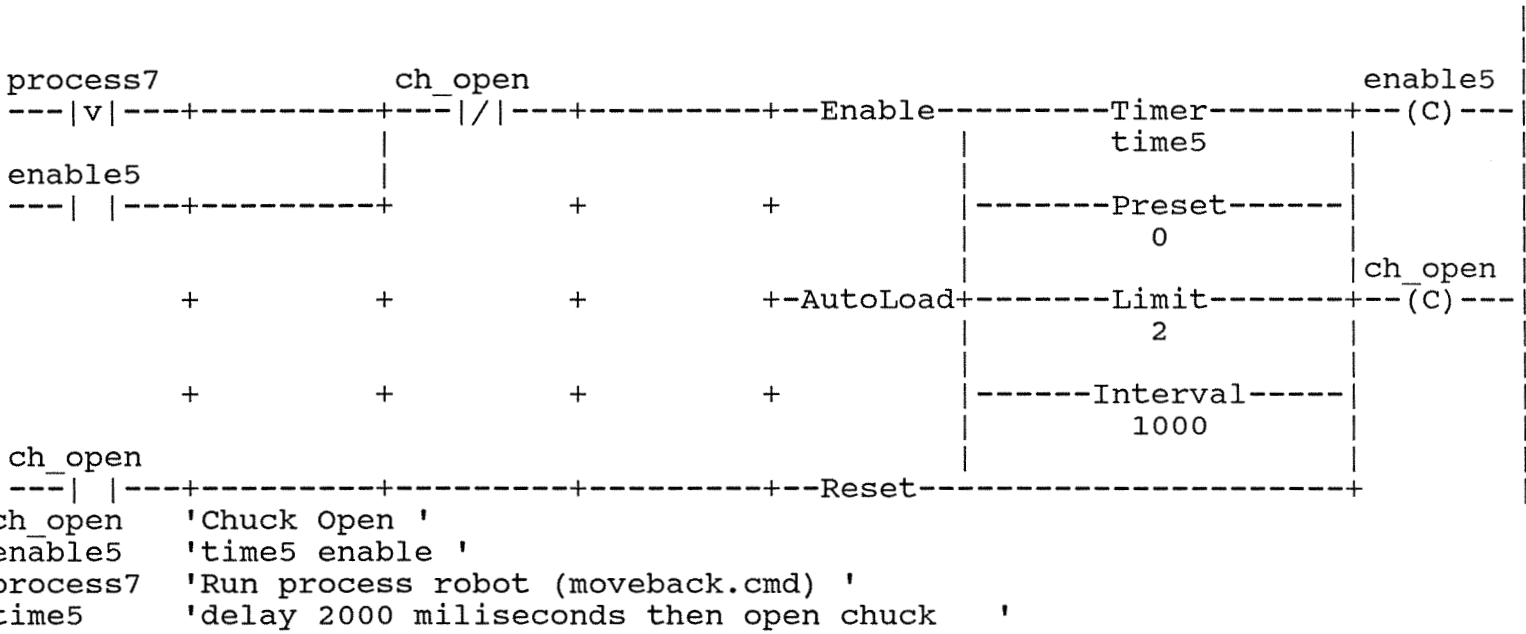
Rung 46

This queue rung start the process that move robot back



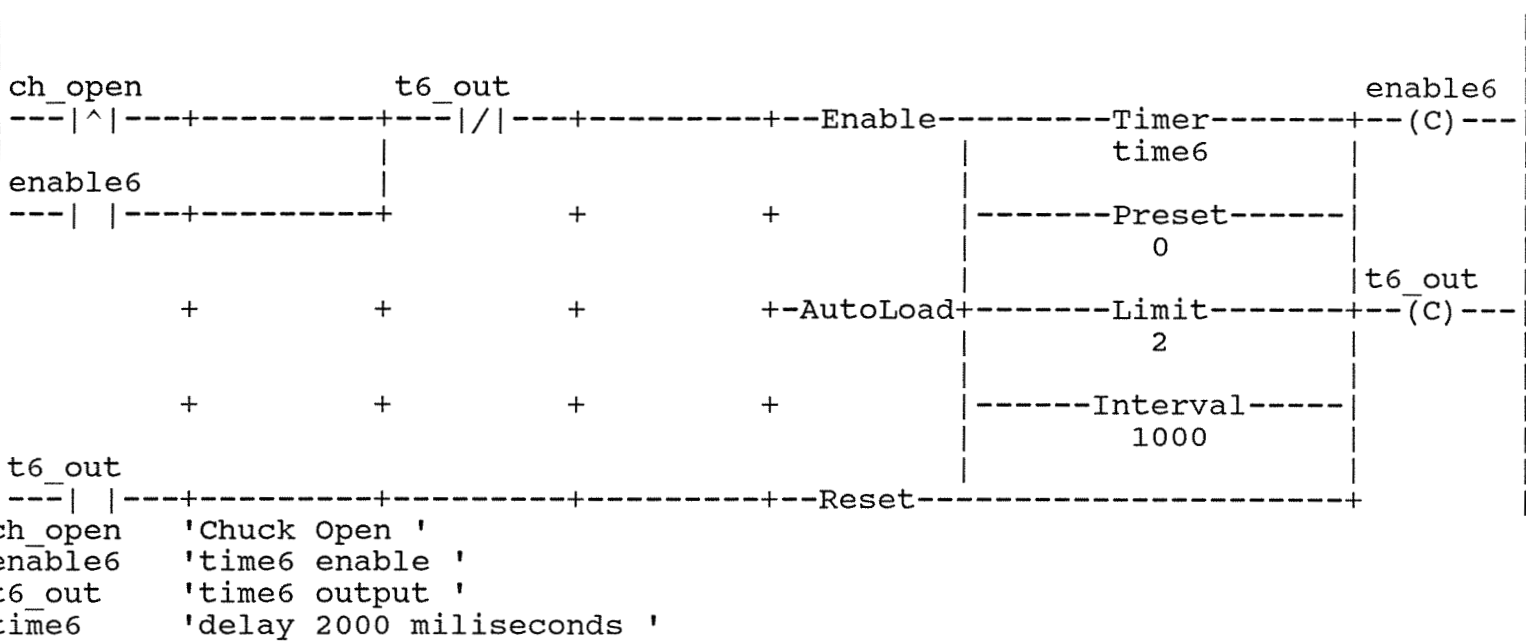
Rung 47

This timer rung delay 2000 milliseconds then open chuck



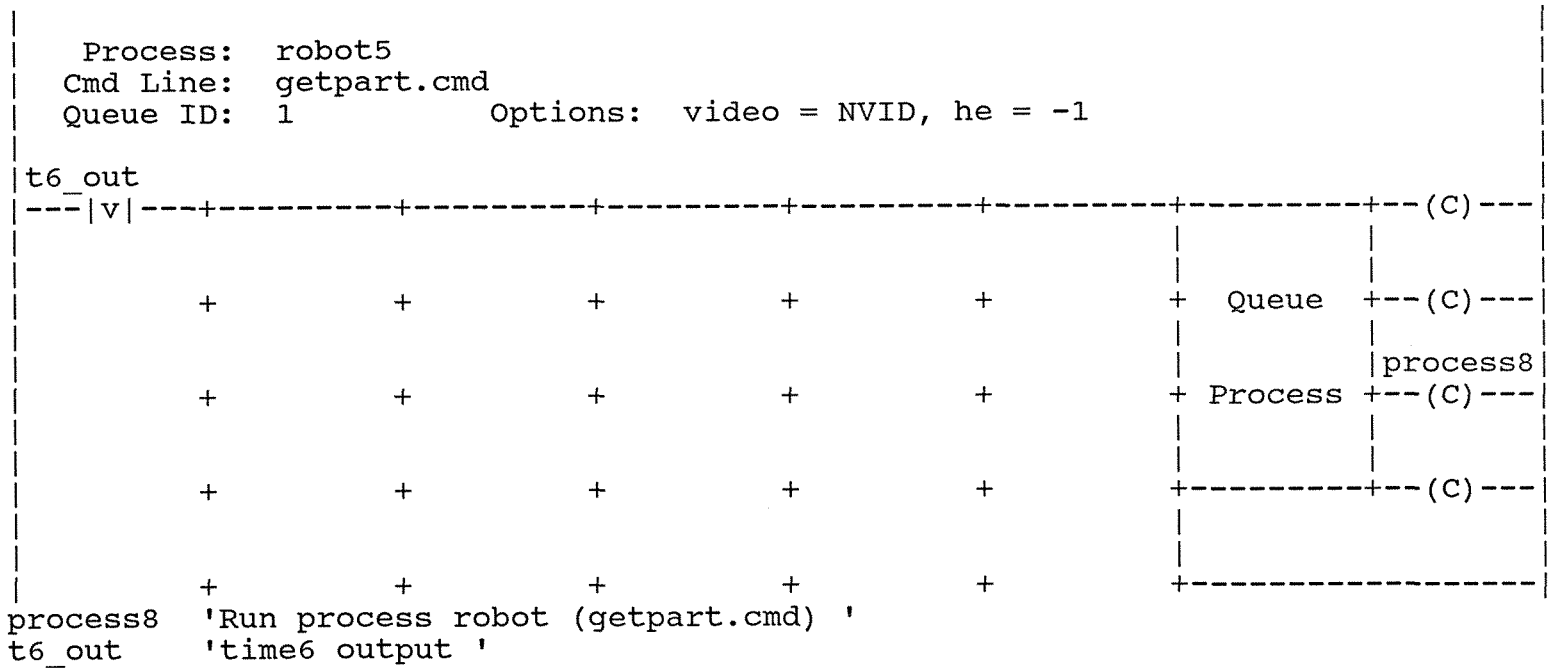
Rung 48

This timer rung delay 2000 milliseconds in order to run robot



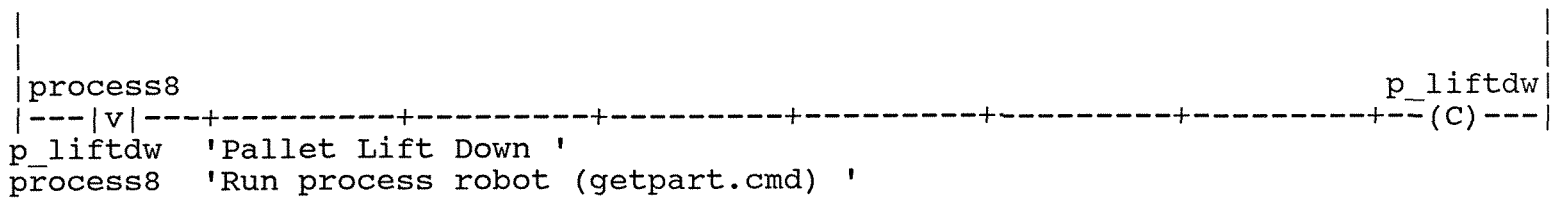
Rung 49

This queue rung start the process in which robot gets part



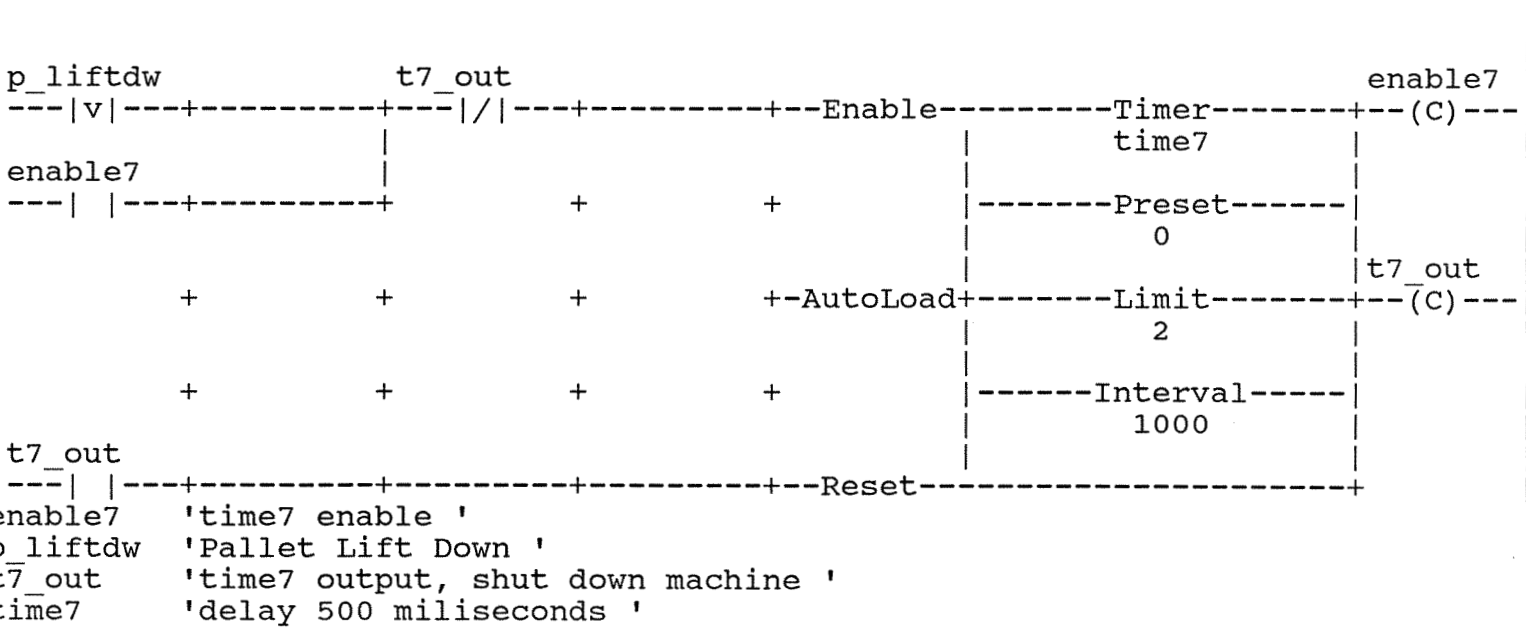
Rung 50

This matrix rung gets pallet down

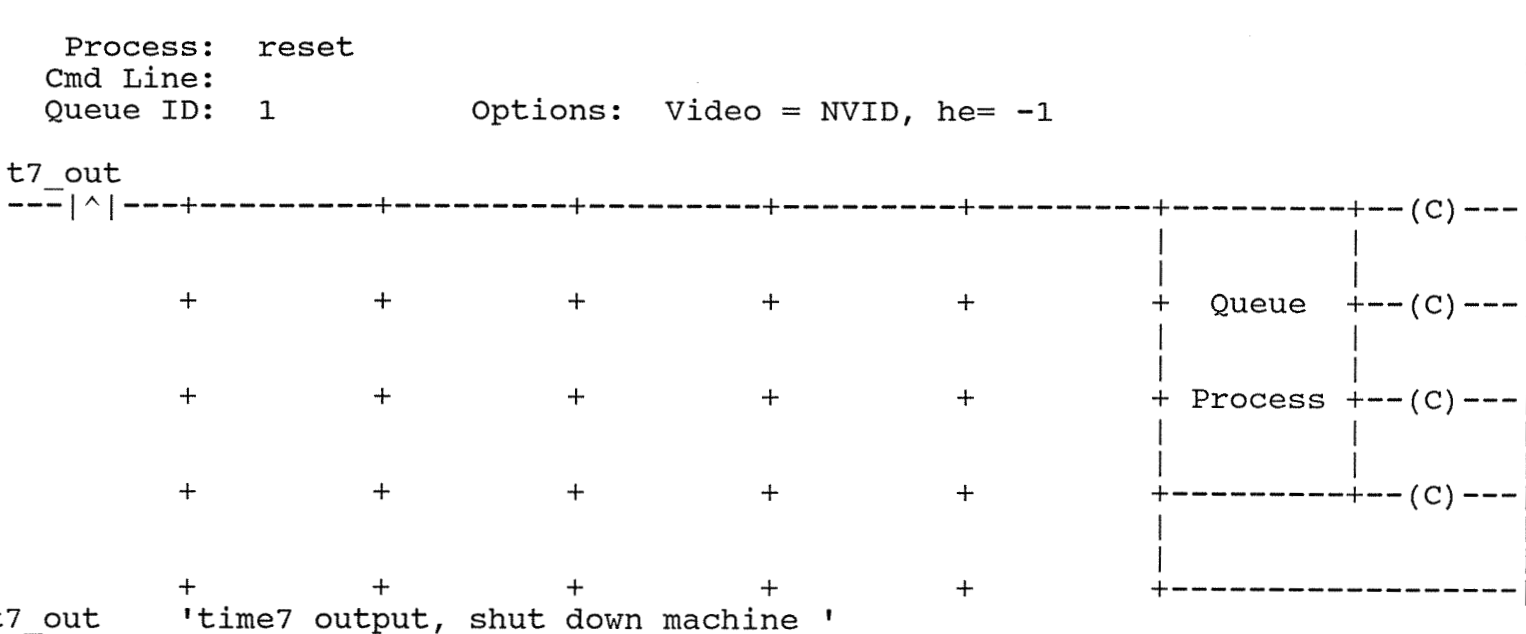


Rung 51

This timer rung delay 2000 milliseconds then shut down machine



Rung 52



TATION\\C:\OC\test6.oc
riday, May 12, 1995 6:06 pm
se OMEGA Controlware to control the CIM cell

adder	Position	Items	Description/Parameters
oot	1 <p1>	52	

ladder found.

```

title 1      Root      Application title <p1>

data 2      Root      Unsigned byte port_a <p1>
              'CIM Cell Control Input Module '
              (cim1)
.4 (la_stop)  'lathe Stop '
              Read      Root      Rung 46 <p10>
.5 (p_lifted) 'Pallet Lifted '
              Read      Root      Rung 39 [P_UP] <p7>,
              Rung 40 <p7>, Rung 41 <p8>
.6 (p_arr)   'Pallet Arrived '
              Read      Root      Rung 35 [P_ON] <p5>,
              Rung 38 [CON_OFF] <p6>,
              Rung 39 [P_UP] <p7>
.7 (pho_cell) 'Photo Cell '
              Read      Root      Rung 36 [P_OFF] <p6>

data 3      Root      Unsigned byte port_b <p1>
              'CIM Cell Control Output Module '
              (cim2)
.0 (la_hand)  'Lathe Handshank '
.1 (la_g66)   'Lathe G66inp '
.2 (la_run)   'lathe Running '

data 4      Root      Unsigned byte port_c <p1>
              'CIM Cell Control Output Module '
              (cim3)
.0 (p_stops)  'Pallet Stops '
              Read      Root      Rung 38 [CON_OFF] <p6>
              Write     Root      Rung 35 [P_ON] <p5>,
              Rung 36 [P_OFF] <p6>
.1 (ch_open)  'Chuck Open '
              Read      Root      Rung 48 <p11>
              Read/write Root    Rung 47 <p11>
              Write     Root      Rung 40 <p7>
.2 (la_start) 'Lathe Start '
              Read/write Root    Rung 45 <p10>
.3 (ch_close) 'Chuck Close '
              Read      Root      Rung 43 <p9>
              Read/write Root    Rung 42 <p8>
.4 (p_liftup) 'Pallet Lift Up '
              Read/write Root    Rung 39 [P_UP] <p7>
.5 (conveyor) 'Conveyor '
              Write     Root      Rung 37 [CON_ON] <p6>,
              Rung 38 [CON_OFF] <p6>
.6 (p_liftdw) 'Pallet Lift Down '
              Read      Root      Rung 35 [P_ON] <p5>,
              Rung 37 [CON_ON] <p6>,
              Rung 51 <p13>
              Write     Root      Rung 50 <p12>

Data 5      Root      Bit process2 <p1> 'Run process robot (nest.cmd) '
              Read/write Root    Rung 32 <p4>
  
```

ata 6	Root	Bit process3 <p2> Read Root Write Root	'Run process lathel ' Rung 33 <p4>, Rung 34 <p5> Rung 32 <p4>
ata 7	Root	Bit process4 <p2>	'Run process poll '
ata 8	Root	Bit process5 <p2> 'Run process robot (loadpart.cmd) ' Read Root Write Root	Rung 42 <p8> Rung 41 <p8>
ata 9	Root	Bit process6 <p2> 'Run process robot (moveaway.cmd) ' Read Root Write Root	Rung 45 <p10> Rung 44 <p9>
ata 10	Root	Bit process7 <p2> 'Run process robot (moveback.cmd) ' Read Root Write Root	Rung 47 <p11> Rung 46 <p10>
ata 11	Root	Bit process8 <p2> Read Root Write Root	'Run process robot (getpart.cmd) ' Rung 50 <p12> Rung 49 <p12>
ata 12	Root	Bit process9 <p2>	'Run process reset '
ata 13	Root	Bit start <p2> Read Root Write Root	'start conveyor running ' Rung 35 [P_ON] <p5>, Rung 37 [CON_ON] <p6> Rung 34 <p5>
ata 14	Root	Bit enable1 <p2> Read/write Root	'time1 enable ' Rung 39 [P_UP] <p7>
ata 15	Root	Bit enable2 <p2> Read/write Root	'time2 enable ' Rung 42 <p8>
ata 16	Root	Bit enable3 <p2> Read/write Root	'time3 enable ' Rung 43 <p9>
ata 17	Root	Bit enable4 <p2> Read/write Root	'time4 enable ' Rung 45 <p10>
ata 18	Root	Bit enable5 <p2> Read/write Root	'time5 enable ' Rung 47 <p11>
ata 19	Root	Bit enable6 <p3> Read/write Root	'time6 enable ' Rung 48 <p11>
ata 20	Root	Bit enable7 <p3> Read Root Read/write Root	'time7 enable ' Rung 39 [P_UP] <p7> Rung 51 <p13>

Data 21	Root	Bit t3_out <p3> 'time3 output ' Read Root Rung 44 <p9> Read/write Root Rung 43 <p9>
Data 22	Root	Bit t6_out <p3> 'time6 output ' Read Root Rung 49 <p12> Read/write Root Rung 48 <p11>
Data 23	Root	Bit t7_out <p3> 'time7 output, shut down machine ' Read Root Rung 36 [P_OFF] <p6>, Rung 38 [CON_OFF] <p6>, Rung 52 <p13> Read/write Root Rung 51 <p13>
Data 24	Root	New <p3>
Data 25	Root	Signed word time1 <p3> 'Delay 1000 milliseconds then lift Pallet up ' Read/write Root Rung 39 [P_UP] <p7>
Data 26	Root	Signed word time2 <p3> 'delay 1000 milliseconds then get chuck close ' Read/write Root Rung 42 <p8>
Data 27	Root	Signed word time3 <p3> 'delay 2000 milliseconds ' Read/write Root Rung 43 <p9>
Data 28	Root	Signed word time4 <p3> 'delay 2000 milliseconds then start lathe ' Read/write Root Rung 45 <p10>
Data 29	Root	Signed word time5 <p3> 'delay 2000 milliseconds then open chuck ' Read/write Root Rung 47 <p11>
Data 30	Root	Signed word time6 <p3> 'delay 2000 milliseconds ' Read/write Root Rung 48 <p11>
Data 31	Root	Signed word time7 <p3> 'delay 500 milliseconds ' Read/write Root Rung 51 <p13>
Rung 32	Root	Queue rung <p4>
Rung 33	Root	Queue rung <p4>
Rung 34	Root	Queue rung <p5>
Rung 35	Root	Matrix rung P_ON <p5>
Rung 36	Root	Matrix rung P_OFF <p6>
Rung 37	Root	Matrix rung CON_ON <p6>

```

rung 38      Root      Matrix rung CON_OFF <p6>
rung 39      Root      Timer rung P_UP <p7>
rung 40      Root      Matrix rung <p7>
rung 41      Root      Queue rung <p8>
rung 42      Root      Timer rung <p8>
rung 43      Root      Timer rung <p9>
rung 44      Root      Queue rung <p9>
rung 45      Root      Timer rung <p10>
rung 46      Root      Queue rung <p10>
rung 47      Root      Timer rung <p11>
rung 48      Root      Timer rung <p11>
rung 49      Root      Queue rung <p12>
rung 50      Root      Matrix rung <p12>
rung 51      Root      Timer rung <p13>
rung 52      Root      Queue rung <p13>
  
```

Item Type	Good	Bad	Total
Data	30	0	30
Rung	21	0	21
Graph	0	0	0
memo/title	1	0	1
Configuration	0	0	0
Total	52	0	52


```
h_close      Alias for port_c.3; defined in Data 4 <p1>
port_c      Unsigned byte 'CIM Cell Control Output Module '
            (cim3)
            .3 (ch_close)  'Chuck Close '
                Read      Root      Rung 43 <p9>
                Read/write Root      Rung 42 <p8>

h_open      Alias for port_c.1; defined in Data 4 <p1>
port_c      Unsigned byte 'CIM Cell Control Output Module '
            (cim3)
            .1 (ch_open)  'Chuck Open '
                Read      Root      Rung 48 <p11>
                Read/write Root      Rung 47 <p11>
                Write     Root      Rung 40 <p7>

cim1       Alias for port_a; defined in Data 2 <p1>
port_a     Unsigned byte 'CIM Cell Control Input Module '
            (cim1)
            .4 (la_stop)  'lathe Stop '
                Read      Root      Rung 46 <p10>
            .5 (p_lifted) 'Pallet Lifted '
                Read      Root      Rung 39 [P_UP] <p7>,
                Rung 40 <p7>, Rung 41 <p8>
            .6 (p_arr)    'Pallet Arrived '
                Read      Root      Rung 35 [P_ON] <p5>,
                Rung 38 [CON_OFF] <p6>,
                Rung 39 [P_UP] <p7>
            .7 (pho_cell) 'Photo Cell '
                Read      Root      Rung 36 [P_OFF] <p6>

cim2       Alias for port_b; defined in Data 3 <p1>
port_b     Unsigned byte 'CIM Cell Control Output Module '
            (cim2)
            .0 (la_hand)  'Lathe Handshank '
            .1 (la_g66)  'Lathe G66inp '
            .2 (la_run)  'lathe Running '
```

```

cim3      Alias for port_c; defined in Data 4 <p1>
port_c    Unsigned byte 'CIM Cell Control Output Module '
          (cim3)
.0  (p_stops)  'Pallet Stops '
          Read      Root      Rung 38 [CON_OFF] <p6>
          Write     Root      Rung 35 [P_ON] <p5>,
          Rung 36 [P_OFF] <p6>
.1  (ch_open)  'Chuck Open '
          Read      Root      Rung 48 <p11>
          Read/write Root      Rung 47 <p11>
          Write     Root      Rung 40 <p7>
.2  (la_start) 'Lathe Start '
          Read/write Root      Rung 45 <p10>
.3  (ch_close) 'Chuck Close '
          Read      Root      Rung 43 <p9>
          Read/write Root      Rung 42 <p8>
.4  (p_liftup) 'Pallet Lift Up '
          Read/write Root      Rung 39 [P_UP] <p7>
.5  (conveyor) 'Conveyor '
          Write     Root      Rung 37 [CON_ON] <p6>,
          Rung 38 [CON_OFF] <p6>
.6  (p_liftdw) 'Pallet Lift Down '
          Read      Root      Rung 35 [P_ON] <p5>,
          Rung 37 [CON_ON] <p6>,
          Rung 51 <p13>
          Write     Root      Rung 50 <p12>
  
```

```

CON_OFF   Defined in Rung 38 <p6>
          Matrix rung
  
```

```

CON_ON    Defined in Rung 37 <p6>
          Matrix rung
  
```

```

conveyor  Alias for port_c.5; defined in Data 4 <p1>
port_c    Unsigned byte 'CIM Cell Control Output Module '
          (cim3)
.5  (conveyor) 'Conveyor '
          Write     Root      Rung 37 [CON_ON] <p6>,
          Rung 38 [CON_OFF] <p6>
  
```

```

enable1   Defined in Data 14 <p2>
          Bit      'time1 enable '
          Read/write Root      Rung 39 [P_UP] <p7>
  
```

```

enable2   Defined in Data 15 <p2>
          Bit      'time2 enable '
          Read/write Root      Rung 42 <p8>
  
```

```

enable3   Defined in Data 16 <p2>
          Bit      'time3 enable '
          Read/write Root      Rung 43 <p9>
  
```

enable4	Defined in Data 17 <p2> Bit 'time4 enable ' Read/write Root	Rung 45 <p10>
enable5	Defined in Data 18 <p2> Bit 'time5 enable ' Read/write Root	Rung 47 <p11>
enable6	Defined in Data 19 <p3> Bit 'time6 enable ' Read/write Root	Rung 48 <p11>
enable7	Defined in Data 20 <p3> Bit 'time7 enable ' Read Root Read/write Root	Rung 39 [P_UP] <p7> Rung 51 <p13>
la_g66 port_b	Alias for port_b.1; defined in Data 3 <p1> Unsigned byte 'CIM Cell Control Output Module ' (cim2) .1 (la_g66) 'Lathe G66inp '	
la_hand port_b	Alias for port_b.0; defined in Data 3 <p1> Unsigned byte 'CIM Cell Control Output Module ' (cim2) .0 (la_hand) 'Lathe Handshank '	
la_run port_b	Alias for port_b.2; defined in Data 3 <p1> Unsigned byte 'CIM Cell Control Output Module ' (cim2) .2 (la_run) 'lathe Running '	
la_start port_c	Alias for port_c.2; defined in Data 4 <p1> Unsigned byte 'CIM Cell Control Output Module ' (cim3) .2 (la_start) 'Lathe Start ' Read/write Root	Rung 45 <p10>
la_stop port_a	Alias for port_a.4; defined in Data 2 <p1> Unsigned byte 'CIM Cell Control Input Module ' (cim1) .4 (la_stop) 'lathe Stop ' Read Root	Rung 46 <p10>
p_arr port_a	Alias for port_a.6; defined in Data 2 <p1> Unsigned byte 'CIM Cell Control Input Module ' (cim1) .6 (p_arr) 'Pallet Arrived ' Read Root	Rung 35 [P_ON] <p5>, Rung 38 [CON_OFF] <p6>, Rung 39 [P_UP] <p7>

```
_liftdw      Alias for port_c.6; defined in Data 4 <p1>
_port_c      Unsigned byte 'CIM Cell Control Output Module '
              (cim3)
.6 (p_liftdw) 'Pallet Lift Down '
              Read          Root          Rung 35 [P_ON] <p5>,
              Write        Root          Rung 37 [CON_ON] <p6>,
              Rung 51 <p13>
              Rung 50 <p12>

_lifted      Alias for port_a.5; defined in Data 2 <p1>
_port_a      Unsigned byte 'CIM Cell Control Input Module '
              (cim1)
.5 (p_lifted) 'Pallet Lifted '
              Read          Root          Rung 39 [P_UP] <p7>,
              Rung 40 <p7>, Rung 41 <p8>

_liftup      Alias for port_c.4; defined in Data 4 <p1>
_port_c      Unsigned byte 'CIM Cell Control Output Module '
              (cim3)
.4 (p_liftup) 'Pallet Lift Up '
              Read/write  Root          Rung 39 [P_UP] <p7>

'_OFF        Defined in Rung 36 <p6>
              Matrix rung

'_ON         Defined in Rung 35 <p5>
              Matrix rung

'_stops      Alias for port_c.0; defined in Data 4 <p1>
_port_c      Unsigned byte 'CIM Cell Control Output Module '
              (cim3)
.0 (p_stops)  'Pallet Stops '
              Read          Root          Rung 38 [CON_OFF] <p6>
              Write        Root          Rung 35 [P_ON] <p5>,
              Rung 36 [P_OFF] <p6>

'_UP         Defined in Rung 39 <p7>
              Timer rung

pho_cell     Alias for port_a.7; defined in Data 2 <p1>
_port_a      Unsigned byte 'CIM Cell Control Input Module '
              (cim1)
.7 (pho_cell) 'Photo Cell '
              Read          Root          Rung 36 [P_OFF] <p6>
```

```
port_a      Defined in Data 2 <p1>
            Unsigned byte 'CIM Cell Control Input Module '
            (cim1)
.4 (la_stop)  'lathe Stop '
            Read          Root          Rung 46 <p10>
.5 (p_lifted) 'Pallet Lifted '
            Read          Root          Rung 39 [P_UP] <p7>,
            Rung 40 <p7>, Rung 41 <p8>
.6 (p_arr)    'Pallet Arrived '
            Read          Root          Rung 35 [P_ON] <p5>,
            Rung 38 [CON_OFF] <p6>,
            Rung 39 [P_UP] <p7>
.7 (pho_cell) 'Photo Cell '
            Read          Root          Rung 36 [P_OFF] <p6>

port_b      Defined in Data 3 <p1>
            Unsigned byte 'CIM Cell Control Output Module '
            (cim2)
.0 (la_hand)  'Lathe Handshank '
.1 (la_g66)   'Lathe G66inp '
.2 (la_run)   'lathe Running '

port_c      Defined in Data 4 <p1>
            Unsigned byte 'CIM Cell Control Output Module '
            (cim3)
.0 (p_stops)  'Pallet Stops '
            Read          Root          Rung 38 [CON_OFF] <p6>
            Write         Root          Rung 35 [P_ON] <p5>,
            Rung 36 [P_OFF] <p6>
.1 (ch_open)  'Chuck Open '
            Read          Root          Rung 48 <p11>
            Read/write    Root          Rung 47 <p11>
            Write         Root          Rung 40 <p7>
.2 (la_start) 'Lathe Start '
            Read/write    Root          Rung 45 <p10>
.3 (ch_close) 'Chuck Close '
            Read          Root          Rung 43 <p9>
            Read/write    Root          Rung 42 <p8>
.4 (p_liftup) 'Pallet Lift Up '
            Read/write    Root          Rung 39 [P_UP] <p7>
.5 (conveyor) 'Conveyor '
            Write         Root          Rung 37 [CON_ON] <p6>,
            Rung 38 [CON_OFF] <p6>
.6 (p_liftdw) 'Pallet Lift Down '
            Read          Root          Rung 35 [P_ON] <p5>,
            Rung 37 [CON_ON] <p6>,
            Rung 51 <p13>
            Write         Root          Rung 50 <p12>

process2    Defined in Data 5 <p1>
            Bit          'Run process robot (nest.cmd) '
            Read/write   Root          Rung 32 <p4>
```

```
rocess3      Defined in Data 6 <p2>
Bit          'Run process lathe1 '
             Read      Root      Rung 33 <p4>, Rung 34 <p5>
             Write     Root      Rung 32 <p4>

rocess4      Defined in Data 7 <p2>
Bit          'Run process poll '

rocess5      Defined in Data 8 <p2>
Bit          'Run process robot (loadpart.cmd) '
             Read      Root      Rung 42 <p8>
             Write     Root      Rung 41 <p8>

rocess6      Defined in Data 9 <p2>
Bit          'Run process robot (moveaway.cmd) '
             Read      Root      Rung 45 <p10>
             Write     Root      Rung 44 <p9>

rocess7      Defined in Data 10 <p2>
Bit          'Run process robot (moveback.cmd) '
             Read      Root      Rung 47 <p11>
             Write     Root      Rung 46 <p10>

rocess8      Defined in Data 11 <p2>
Bit          'Run process robot (getpart.cmd) '
             Read      Root      Rung 50 <p12>
             Write     Root      Rung 49 <p12>

rocess9      Defined in Data 12 <p2>
Bit          'Run process reset '

start        Defined in Data 13 <p2>
Bit          'start conveyor running '
             Read      Root      Rung 35 [P_ON] <p5>,
             Write     Root      Rung 37 [CON_ON] <p6>
             Write     Root      Rung 34 <p5>

t3_out       Defined in Data 21 <p3>
Bit          'time3 output '
             Read      Root      Rung 44 <p9>
             Read/write Root      Rung 43 <p9>

t6_out       Defined in Data 22 <p3>
Bit          'time6 output '
             Read      Root      Rung 49 <p12>
             Read/write Root      Rung 48 <p11>

t7_out       Defined in Data 23 <p3>
Bit          'time7 output, shut down machine '
             Read      Root      Rung 36 [P_OFF] <p6>,
             Write     Root      Rung 38 [CON_OFF] <p6>,
             Write     Root      Rung 52 <p13>
             Read/write Root      Rung 51 <p13>
```

.me1 Defined in Data 25 <p3>
Signed word 'Delay 1000 milliseconds then lift Pallet up '
 Read/write Root Rung 39 [P_UP] <p7>

.me2 Defined in Data 26 <p3>
Signed word 'delay 1000 milliseconds then get chuck close '
 Read/write Root Rung 42 <p8>

.me3 Defined in Data 27 <p3>
Signed word 'delay 2000 milliseconds '
 Read/write Root Rung 43 <p9>

.me4 Defined in Data 28 <p3>
Signed word 'delay 2000 milliseconds then start lathe '
 Read/write Root Rung 45 <p10>

.me5 Defined in Data 29 <p3>
Signed word 'delay 2000 milliseconds then open chuck '
 Read/write Root Rung 47 <p11>

.me6 Defined in Data 30 <p3>
Signed word 'delay 2000 milliseconds '
 Read/write Root Rung 48 <p11>

.me7 Defined in Data 31 <p3>
Signed word 'delay 500 milliseconds '
 Read/write Root Rung 51 <p13>

1 names in report.