The Practice of Industrial Logic Design [1]

M. R. Lucas and D. M. Tilbury
Engineering Research Center for Reconfigurable Machining Systems
Department of Mechanical Engineering
University of Michigan, Ann Arbor, MI 48109-2125
{mrlucas, tilbury}@umich.edu

Abstract

Many academic researchers have been working on the problem of how to improve industrial logic design. The problem that many are trying to solve is the perceived inefficiency of the current methods, which use primitive, low-level design languages, practically no logic reuse, and are very time consuming. To solve these problems researchers have focused on methods which can be verified against a known specification language, or which can be automatically generated from a specification. This work has generally been done with a minimal understanding of what the current logic design methods actually are.

In this work, we present the results of an observational study of the current methods of creating control logic. We find that the current specifications are generally informal and loosely defined, and that the typical logic designer is responsible for determining the details of system behavior, anticipating potential problems, and coordinating with other designers. This is a larger range of activities than generally addressed by logic design schemes focused on verification or automatic logic generation.

1 Introduction

Control logic is a critical part of a modern machining system. The control logic is responsible for insuring the machine operates in a safe and productive manner, coordinating tens of thousands of I/O points.

In the US, the most popular method of creating control logic is ladder diagrams (see example in fig. 1). This is one of the five languages specified in the IEC 61131-3 standard of control languages for machining systems [5]. Ladder logic began as a systematic method of laying out physical relays to control machines before microprocessors were common. Modern implementations of ladder diagrams use a PC to write the logic, and the "relays" are simulated in a special purpose computer called a Programmable Logic Controller (PLC). This has saved a great deal of time compared to physically wiring thousands of relays.

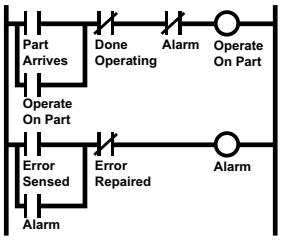

Figure 1: A trivial program written in ladder diagrams. The `Operate On Part` output will be set when a `Part Arrives`, and remain set until either `Done Operating` or `Alarm`. The `Alarm` will be set when `Error Sensed` and remain on until `Error repaired`.

However, the system still needs to be improved. Currently there is a demand for machines to be produced faster and for the control features on these machines to be more sophisticated. However, logic designers are not given enough tools to effectively satisfy this demand, and are often not thoroughly trained to use the features that they do have.

With this in mind we have performed an observational study of current current logic design practices at Lamb Technicon. From September to December of 2001, logic designers were observed for about 110 hours while working on three separate projects.

The rest of the paper is organized as follows: section 2 reviews literature in this field; section 3 reviews the methods used in this study; section 4 presents the main results; and section 5 discusses many of the more intangible observations made.

2 Relevant Literature

The industrial specification IEC 61131-3 [5] contains five programming languages which may be supported by any compliant vendor. These are: instruction list (similar to assembly), structured text (similar to For-

[1] This research was supported in part by the NSF under grant EEC95-92125.

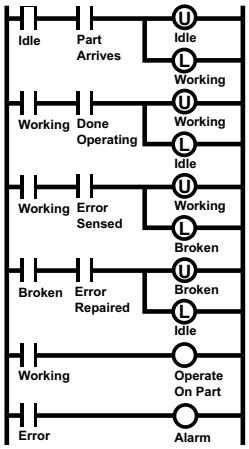

Figure 2: A simple ladder program using "token-passing logic". The symbols – (L) – and – (U) – represent latch and unlatch coils, which only alter the state of a bit if the rung evaluates as true. Compare to the example in figure 1.

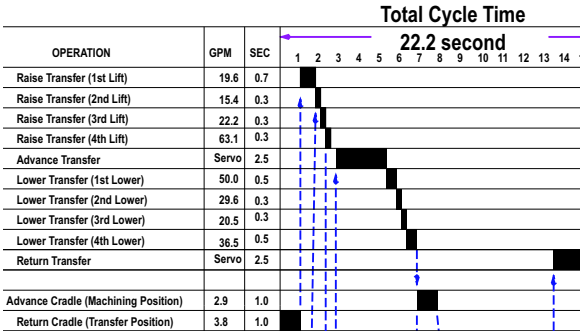

Figure 3: A portion of a timing bar chart, which is used to specify desired automatic mode behavior for a machine.

tran), function block diagrams (an example of data flow graphs), sequential function charts (a simplified version of Petri nets), and ladder diagrams (similar to electrical relay diagrams). Most industrial logic design solutions rely on one of these languages with minimal support for other languages. In the United States ladder diagrams are the most common method of creating control logic for industrial problems.

To address the problems of ladder logic without abandoning the ladder framework, Ponizil [13] suggested applying structured programming techniques, including modularity and top-down design, to ladder diagrams. A similar method was proposed by Morihara [11], although we have not seen any evidence of these techniques being used in practice. Other researchers have generated ladder diagrams from Petri nets in order to utilize formal design methods with existing, ladder based, industrial hardware (see [15]). This line of research generally uses a variation of "token-passing logic" which creates one rung for each transition in the Petri net and used latched coils to maintain state (see example in fig. 2). This is in contrast to ladder diagrams written by logic designers which generally do not use latched coils, and use one rung per output. Token-passing logic would defeat the primary debugging methods used in industry today.

Petri nets are often used as an alternative logic control design methodology. Park *et al.* [12] have developed methods of directly converting timing bar charts into logic written in Petri nets (see fig. 3 for a sample timing bar chart). Frey *et al.* [10] describe "signal interpreted Petri nets" (SIPNs). They perform an experiment determining that SIPNs are easier to generate than function block diagrams, and that the computerized aids which are possible in SIPNs prevent many errors. Petri nets are also assumed to be the solution in other papers. These include Uzam *et al.* [15] who create a controller for a small demo system, Holloway *et al.* [2] who created software to allow Petri nets control systems using a PC, and Lee and Hsu [4] who use Petri nets to design logic, and then convert to ladder diagrams for use with industrial PLCs.

In addition to the work in Petri nets, some work has been done using finite state machines as programming tools. Endsley *et al.* [1, 6] use "Modular Finite State Machines" to create a modular structure for designing controllers, and an example of the system in use is presented in [14].

None of these alternative methodologies have been implemented in industrial scale logic programming.

One thing that is missing from the literature to date is a complete understanding of the problems associated with logic control for machining systems. For example, typical Petri nets used in academic papers have contained 63 places [7, 9], and 21 places [15]. Projects observed during this study contained *tens of thousands* of rungs, and it is not clear that measurements and intuitions developed using small systems will work for large systems.

It is difficult to imagine a cost-effective and expedient method of determining the cost of using a particular logic control design methodology. A first step is to perform a task analysis [3] to determine what is done using the current system.

3 Study Methods

From September to December of 2001, observations were made at Lamb for approximately 110 hours on 28 different days. During this time, portions of three projects in three different stages of development were observed.

The primary project observed was in the middle of the

development cycle. Most of the team leader's time was spent coordinating the project among the team members, as well as entering logic from the previous project, and managing the memory map (see table 2).

The the two additional projects are covered in more depth in [8].

During this time approximately 130 pages of notes were taken by hand. The notes included a summary of the activities performed broken down into ten minute intervals, as well as descriptions of subtasks used to complete a single task when possible, and any other relevant observations. Data taken during the cycle and debug stage was taken in 20 minute intervals, due to the less structured nature of the task.

After the observations were complete, each description of subtasks was separated from the notes and typed up. Similar tasks were grouped into categories. Using this method the following activity categories were developed: Project coordination and planning, File creation and maintenance, Memory management, Copy/Modify logic entry, Cognitive logic development, and Debugging (see table 1 for details).

Once the activity categories were developed, the time-based data was examined. Each 10 minute portion of time was categorized into either one of the six project related activities, or an additional non-project related task. If multiple activities were recorded in the ten minute section of time, the primary activity was recorded.

4 Study Results

4.1 Overview of Logic Development Process

A simple description of the approach used to create control logic is shown in figure 4. The logic designers are given project specific specifications and schematics. Using an additional set of standard specifications, usually in the form of a previous project, they create the logic needed to control the machine. Project specific requirements include details about the actions the machine must perform to create parts, diagrams of physical and electrical components, and a description of the diagnostics desired. The standard specifications include the details of implementing the system and also include the needed safety and reliability requirements.

The amount of time and number of people required can vary greatly from project to project. However, a typical project may require 6 months and 4-6 people. A sample timeline is shown in figure 5.

A project is usually to write the logic for one machine. A machine usually consists of one transfer bar and about five or more stations on both sides of the transfer bar (see fig. 6). During a typical cycle, the transfer bar picks up all the parts in the machine, moves them to the

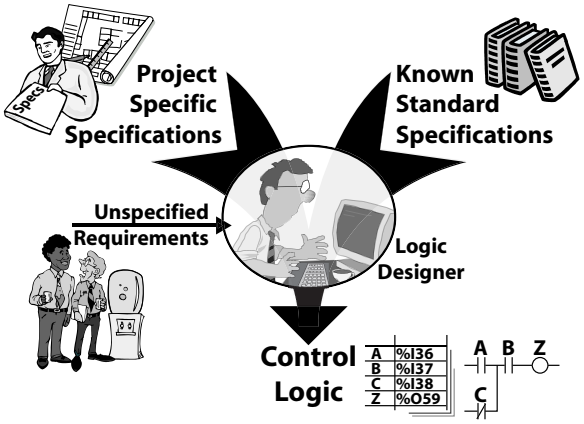

Figure 4: Overview of the logic generation process: Logic Designers are given control specifications and machine schematics to describe the logic needed. These are combined with standard specifications (usually in the form of a previous project) to create the needed control logic. Unspecified requirements can include late changes or unexpected constraints in the machine or electronics.

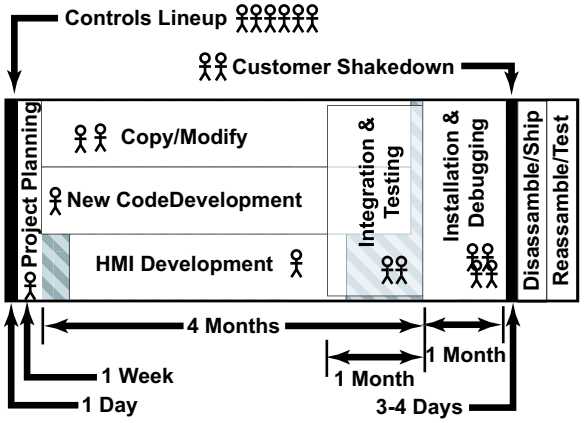

Figure 5: Logic Development Timeline: Example timeline of the development of a control logic project, including both time and manpower required.

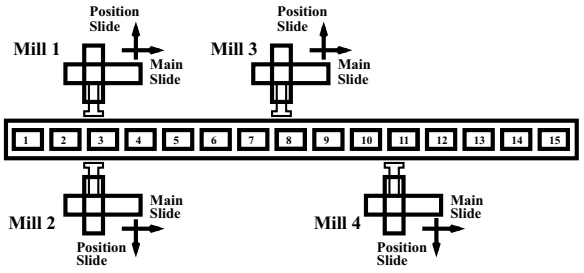

Figure 6: Schematic of a simple machine. This machine contains four cutting stations (one for each mill listed) and more than 13 part positions.

next station, sets all the parts down, and then retracts. While the transfer bar retracts each part is clamped, and then operated on by the appropriate station. Cycle times are generally less than a minute; the number of parts produced is often greater than 200,000/year.

The resulting logic is required to perform many tasks. It must move the machine according to its specifications to create parts. It must also provide any number of safety interlocks, which ensure that the machine will not hurt any operators or itself, even in the presence of operator errors or machine malfunction (see section 5.1). Data used for the human-machine interface (HMI) is maintained; manual and hand (or semi-automatic) modes are created, and are subjected to the same safety interlocks as auto mode. Special purpose modes, such as unusual features required for spindle diagnostics, are added. Finally any data required for diagnostic messages must be created and maintained. The logic to create the automatic mode is generally reported to be about 10% of the total.

4.2 Activities observed

There are several separate activities that are needed to successfully generate industrial logic. While observing the logic designers, most of their activities could be divided into six basic categories: project coordination and documentation, creating and managing files, memory management, copy/modify, cognitive development, and debugging (see table 1). There is no separate category for top-level design, as would be expected in a software development team. Since most of the logic is taken from a previous project, much of the top level design is implied from the start. The rest occurs within the context of either project coordination (e.g. supervisors telling engineers how the project should look in the end), or memory management (e.g. allocating certain blocks of memory to certain people and functions, thereby implying a certain data structure).

A tabulation of time spent in the various activities can be found in table 2. The data only reflects the time spent by lead designers. Additional data from team members and debuggers is in [8].

It is interesting to note how much time the team leaders spend coordinating with their team members. This coordination is mostly discussing and confirming the communication protocols between the various processors needed to make the machine run. For example, in the primary project observed, the main PLC logic interfaced with separate CNC processors which controlled the machine. A lot of time was spent ensuring that the communication between the different processors was consistent.

4.3 Objects used when Developing Logic

In addition to the types of activities, it is important to understand the tools and documents used through the development process. The primary objects used are described below. Their relationship to the activities described in section 4.2 is shown in figure 7.

Table 2: Tabulation of data from *control team leaders*. Time spent in planned meetings was not considered.

Activity	Minutes Observed	Percent of project time
Project Coordination	920	44 %
Cognitive Development	130	6 %
Copy/Modify	450	22 %
Memory Management	310	15 %
File Maintenance	70	3 %
Debugging	200	10 %
GUI Development	0	0 %
Talking to data taker	320	N/A
Break (lunch etc)	370	N/A
Other	320	N/A

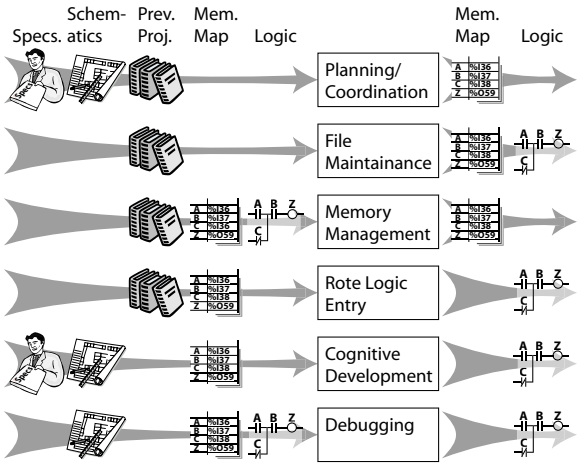

Figure 7: Relationship between design activities and the objects used showing inputs and outputs of each activity.

Project Specifications Formal specifications are provided during the "controls lineup" near the beginning of the project.

Mechanical Drawings Mechanical drawings were used to verify the presence or absence of components, or to verify sensor and actuator locations.

Electrical Drawings Electrical drawings were used during the design phase, to ensure that electrical components were properly interfaced, and during the debugging stage, since it is often difficult to tell a logic error from a wiring error.

Printout of previous project A similar project is usually found to be used as a template for the current project.

The Memory Map This is a map of all the memory, inputs and outputs in the system. It always in-

Table 1: Categorization of activities performed

Project Coordination and Planning	Coordinating between the various people working on a project, planning for a project, or creating documentation for the project. Most coordination is to ensure consistent communications between various processors in the system.
File Creation and Maintenance	Creating new files, performing version control and similar activities.
Memory Management	Creating or modifying the manually allocated memory space of the project.
Copy/Modify	Entering logic by copying from another source. This can be either from a previous project, or from other portions of the current project. This usually includes making minor modifications, such as changing the names on the rungs.
Cognitive Development	Creating new logic. This is usually done for features which were not present on the previous machine.
Debugging	Testing existing logic and making any changes necessary.

cludes the physical address, a descriptive comment. Often included are a retentive check box (i.e. whether it will be stored during power down), and a short name.

The Logic Files These are the primary files which control the machine, akin to the source code in software development.

4.4 Summary of Results

A summary of the logic development process used by the logic design team leader is shown in figure 8. During this time approximately one quarter of his time is spent actually generating logic for the controller. Approximate times for each sub-step of this process are listed in the figure. For a project requiring 3000 rungs from the team leader, this process will take approximately five months, close to the observed time of four months shown in figure 5.

5 Discussion of Observations

5.1 Logic Designers

One of the most striking observations is the expertise of the logic designers, especially the team leaders. The designers are capable of understanding and debugging wiring diagrams; they understand both the machine and the machine users, and often imagine many possible safety issues which would otherwise go unchecked.

The logic designers attempt to consider every possible condition that could occur as they create the control logic. Among the errors and special conditions that they actively considered were:

- Intentional circumvention of the built in safety devices
- System wide power loss at any time
- Processor failure and replacement (the new processor should correctly handle all parts in process, with minimal part loss)

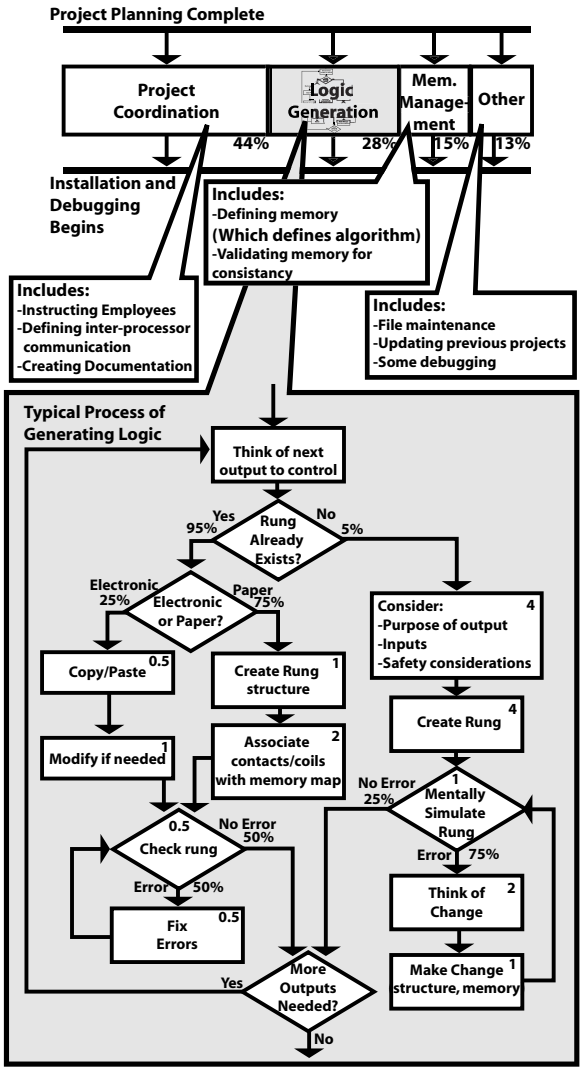

Figure 8: Logic Development Flowchart: Flowchart describing the tasks required of the logic design team leader during a logic design project. Numbers within the block represent an approximate estimate (in minutes) of the time to complete a task.

- Users manually altering the contents of the memory
- Relay failures
- Sensor failures
- Tool breakage

The goal of the logic was to operate the machine as safely and productively as possible under any conceivable condition.

5.2 Ladders and their development environments

The choice of control hardware, development language and development environment are extremely coupled. For example, if an end-user requests that Allen-Bradley control hardware be used, that implies that the project will be developed in ladder logic using Allen-Bradley's RSLogix software. Control vendors are experts in creating a unified control package, they are not experts in creating a usable development environment. This likely adds to the difficulty of using the various development environments.

New logic is typically developed from timing bar charts, which describe the time dependent (sequential) behavior of either the physical machine or the communication signals needed. This is translated in an ad-hoc manner into time-independent (declarative) logic. This mapping is neither consistent from one timing bar chart to another nor easy to determine for a given timing bar chart.

Despite some of the apparent disadvantages described in this paper and others from academia, ladder diagrams have some advantages. For example, it is nearly impossible to create an infinite recursive/iterative loop using ladder diagrams, especially using the methods described here. This means that even if a portion of the logic is poorly written, most of the machine will continue to operate as designed, including safety interlocks. In addition, the primary users of the control logic began their careers as electricians, and the framework of ladder diagrams provides a clear and consistent model of the operations of a complicated computer, without requiring programming. A final point, PLCs don't crash. Designers routinely talk about machines running for years without problems. It is likely that a machine will need to be stopped due to an error in the logic, or an error in the machine, but they almost never stop due to an error in the underlying operating system of the PLC. Any proposal to replace ladder diagrams must preserve as many of these advantages as possible.

6 Acknowledgements

I would like to extend my thanks to Lamb for allowing me access to their designers and their expertise. I would especially like to thank the designers who volunteered to let me peer over their shoulders. It was very enlightening.

References

[1] E. W. Endsley, M. R. Lucas, and D. M. Tilbury. Software tools for verification of modular FSM based logic control for use in reconfigurable machining systems. *Japan–U.S.A. Symposium on Flexible Automation*, 2000.

[2] L. E. Holloway, X. Guan, R. Sundaravadivelu, and J. Ashley, Jr. Automated synthesis and composition of taskblocks for control of manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 30(5), October 2000.

[3] D. E. Kieras. Task analysis and the design of functionality. In A. B. Tucker, editor, *The Computer Science and Engineering Handbook*. CRC Press, 1997.

[4] J. S. Lee and P. L. Hsu. A PLC-based design for the sequence controller in discrete event systems. In *IEEE International Conference on Control Applications*, pages 929–934, September 2000.

[5] R. W. Lewis. *Programming Industrial Control Systems Using IEC 1131-3 Revised Edition*. The Institution of Electrical Engineers, 1998.

[6] M. R. Lucas, E. W. Endsley, and D. M. Tilbury. Coordinated logic control for reconfigurable machine tools. In *Proceedings of the American Control Conference*, pages 2107–2113, 1999.

[7] M. R. Lucas and D. M. Tilbury. Quantative and qualitative comparisons of PLC programs for a small testbed with a focus on human issues. In *Proceedings of the American Control Conference*, May 2002.

[8] M. R. Lucas and D. M. Tilbury. A study of current logic design practices in the automotive manufacturing industry. *International Journal of Human Computer Studies*, 59(5):725–753, November 2003.

[9] M. R. Lucas and D. M. Tilbury. Quantative and qualitative comparisons of PLC programs for a small testbed with a focus on human issues. *International Journal of Advanced Manufacturing Technology*, 2004. Accepted for publication.

[10] M. Minas and G. Frey. Visual PLC-programming using signal interpreted Petri nets. In *Proceedings of the American Control Conference*, pages 5019–5024, 2002.

[11] R. H. Morihara. State-based ladder logic programming. In *Proceedings of the Engineering Society of Detroit*, pages 157–167, Ann Arbor, MI, 1994.

[12] E. Park, D. M. Tilbury, and P. P. Khargonekar. A modeling and analysis methodology for modular logic controllers of machining systems using Petri net formalism. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 31(2):168–188, 2001.

[13] D. Ponizil. Back to basics: The essentials of structured PLC. *Control Engineering*, 48, September 2001.

[14] S. S. Shah, E. W. Endsley, M. R. Lucas, and D. M. Tilbury. Reconfigurable logic control using modular finite state machines: Design, verification, implementation, and integrated error handling. In *Proceedings of the American Control Conference*, 2002.

[15] M. Uzam, A. H. Jones, and I. Yücel. Using a Petri-net-based approach for the real-time supervisory control of an experimental manufacturing system. *International Journal of Advanced Manufacturing Technology*, 16:498–515, 2000.