RTL LOGIC REALIZATION USING
LADDER DIAGRAM FOR PROGRAMMABLE CONTROLLER

Shobha S. 1, S. Ramachandran 2

1 Cambridge Institute of Technology, Bangalore, India
2 SJB Institute of Technology, Bangalore, India

ABSTRACT:

Ladder diagram is a widely used Graphical Programming Language for Programmable Controllers. The design of Programmable Controller using Ladder Diagrams is an experience-based method and verification is typically done through experiments or simulation. This paper aims to design and simulate the attributes used in the Ladder diagram. The proposed method represents each attribute of a Ladder Diagram as a part of the user defined 16-bit Instruction Code. An algorithm for various attributes for efficient realization of Ladder Diagrams has been developed. RTL code has also been designed for the attributes of the Ladder diagram using the developed algorithm. This paper also describes the memory design like Code-memory, Stack-memory and Image-memory. The complete design has been validated by simulation using Modelsim.

Keywords: Register Transfer Level, Ladder Diagram, Attributes, Verilog, ASM Chart, Architecture

[1] INTRODUCTION

Programmable Controllers are widely used in many industrial applications, especially for manufacturing systems due to its ease of operation and powerful functionality. User Application Programs for Programmable Controllers can be written in one or more programming languages which are standardized by International Standards like IEC 61131-3 [1]. Many research outputs have been reported on all five programming languages like Ladder Diagram, Sequential Flow Chart, Functional Block Diagram, Structured Text and Instruction List using IEC 61131-3 Standard.

Over the past decades, Programmable Controllers have evolved from hardwired controls to Ladder Diagram based programming which is more advantageous and a very popular way for programming Programmable Controllers. Ladder Diagram programming is based on electrical wiring diagrams. The basic programming skills needed to develop the applications using Ladder Diagram programs can be learnt relatively quickly and the graphical presentation can be understood almost intuitively, especially suited for Plant technicians. The technique is very easy to understand by any people who are familiar with simple electronic or electrical circuits using symbols of components like relays and switches. Consequently, it is well accepted by operators, electricians and plant technicians. Over the years, researchers have developed many design methods based on Ladder Diagram for implementing control systems. In this section, we review existing techniques of Ladder diagram applications. Kurapati Venkatesh et al. [7] proposed a methodology to evaluate the complexity of Petri Nets and Ladder diagrams for sequence controller design. Complexity of a design is characterized by the number of basic elements used to model the given control logic. Hyung Seok Kim et al. [8] described detailed steps of the translation method that converts a Ladder Diagram directly to a native code for Programmable Logic Controllers used in most automation systems. Norimasa Shoji et al. [9] proposed a control sequence extraction method to translate ladder diagram to Sequential Flow Chart. A tentative Sequential Flow Chart is created by searching all possible states by using transitions in the ladder diagram. Jin-Shyan Lee et al. [10] proposed a new approach via the “If-then” transformation in which both the Ladder Logic Diagram and Petri Net are transformed into the same “If-then” format and evaluate the Ladder Logic Diagram and Petri Net based on the number of “If-then” rules that is defined.

[2] RELATED WORK

Italia Jimenez et al. [11] described the approach that allows to rapidly synthesizing correct programs for Programmable Logic Controller which is based on a set of simple translation rules that produces Ladder diagram from a Timed Interpreted Petri Nets. T. Suesut et al. [12] proposed the transformation structures from Petri Nets to Ladder diagram and possible implementation to control the process through Programmable Controller inputs and outputs. This work mentions that a Petri Net has two types of nodes, namely places and transitions. Lan-Hua et al. [13] introduced an activity of using the six stages of computerized Ladder diagram learning design. Liuwen Huang et al. [14] proposed a method of transformation from PLC ladder Diagram to structured text. In this algorithm, Virtual node is introduced, and then it is combined to establish Activity on Vertex diagraph. Nodes
are treated according to their types and the topological sorting of the Activity on Vertex diaigraph; subsequently the transformation from Ladder diagram to Structured Text is achieved.

LI Dandan et al. [15] proposed a method for inter conversion between Ladder Diagram and Instruction List. This algorithm is based on forest and binary tree. At first the algorithm maps ladder diagram to forest, and then it establishes a binary tree to represent ladder diagram logical relationship with forest. The algorithm realizes the exchange of Programmable Logic Controller ladder diagram and instruction list by traversing binary tree. Zhiyuan Tang et al. [16] proposed a method of Compiling Ladder Diagram based on Node Method. This approach presents node method to compile it in the VC++ environment. The node method eliminates the traditional compilation process into the Instruction List. Using the node method in Ladder Diagram, compiling can make complex series-parallel relationship of Ladder Diagram into a concise relationship between the contacts and the nodes. Xuekun Chen et al. [17] proposed an algorithm for translating the Ladder Diagrams to ordinary Petri Nets. In this approach, Petri net theory is used to simulate and analyze Ladder Diagram programs to check whether Programmable Logic Controller systems are live, reversible and free of race and deadlocks. Qin Xingguo [18] presented a Model Integrated computing based tool prototype for designing the PLCs considering a printing control system as a case study. In this approach, non Ladder Diagram parts of the printing control system are compiled into Structured Text. Yogesh Godhwani et al. [19] described an Automation technique for the protection of Induction Motor using Programmable Logic Controller to mitigate overvoltage, over current, over temperature, over speed and in rush current problems.

This paper proposes a novel method in which the algorithm is developed for various attributes of the Ladder Diagram. Based on this algorithm, the RTL code for the attributes of the Ladder Diagram is designed and simulated. The rest of the paper is organized as follows. Section 3 outlines the Proposed Algorithm for Realization of Logic using Ladder Diagrams. It also describes contacts and coils, attributes, and logical structures used for programming the Ladder Diagrams. Section 4 describes the design of Stack memory, Code memory, Image memory and Attributes used in executing the codes defined for the Ladder Diagrams. Section 5 presents the Simulation results. Conclusion is presented in the last section.

[3] PROPOSED ALGORITHM FOR REALIZATION OF LOGIC USING LADDER DIAGRAMS

Ladder Diagram, or simply Ladder, is a graphical language based on relay circuits used in industrial automation. Each Ladder Diagram is written using two vertical lines that represent the ‘power rails’. These power rails are connected by horizontal lines called ‘rung’. A rung is an electric circuit representing an operation of the Ladder Diagram program. A Ladder Diagram is conventionally read from left to right and from top to bottom. Ladder Logic is in fact a rule-based language, instead of a procedure based language. A “rung” in the ladder consists of two types of contacts. They are Normally Open and Normally Closed contacts. Normally Open contacts will be open (non-conducting) when the input is not energized and closed (conducting) when the input is energized. The
output in the Ladder Diagram is represented by a circle (CR 57) as shown in Fig. 1. If the inputs are open or closed in the correct combination, the power can flow from the ‘Hot rail’ through the inputs to power the output and finally to the ‘Neutral rail’. An input can come from a sensor, switch, or any other component. An output will be a device which may be a switch, Lamp, Relay or any other actuator. A Ladder Diagram is made up of power rails, link elements, contacts and coils [20]. These elements of the Ladder Diagram are shown in Table 1.

Fig. 1 is an example of a ladder logic diagram. To understand this diagram, imagine that the Power Line is connected to the vertical line to the left hand side-marked as “Hot rail”. To the right hand side is the “Neutral rail”. In the figure, there are three rungs. In the first rung, there are two normally open contacts marked (1, 2) and the second rung contains four normally open contacts (3 to 6). In the proposed algorithm, the attributes that are defined for programming the Ladder Diagram are illustrated in Table 2. These attributes are stored in the Code memory in the form of 16-bit user defined code, subsequently fetched and executed.

In Table 2, logic diagrams of each attribute represent the way the ladder diagram is programmed. The logic structures that are mainly used in a Ladder Diagram are illustrated in Table 3. It shows the way the attributes are connected to form the Ladder diagram.

**Table 1 Contacts and Coil**

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1" alt="Symbol" /></td>
<td>Normally Open contacts (switch, relay, other ON/OFF devices)</td>
</tr>
<tr>
<td>2</td>
<td><img src="image2" alt="Symbol" /></td>
<td>Normally Closed contact (switch, relay, etc.)</td>
</tr>
<tr>
<td>3</td>
<td><img src="image3" alt="Symbol" /></td>
<td>Coil (output loads: motor, lamp, solenoid, etc.)</td>
</tr>
</tbody>
</table>

**Table 2 Attributes of Ladder Diagram**

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Attributes</th>
<th>Ladder Logic Diagrams</th>
<th>Attribute Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nil (NO)</td>
<td><img src="image4" alt="Diagram" /></td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>OPEN</td>
<td><img src="image5" alt="Diagram" /></td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>RETURN</td>
<td><img src="image6" alt="Diagram" /></td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>UP</td>
<td><img src="image7" alt="Diagram" /></td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>OPEN/UP</td>
<td><img src="image8" alt="Diagram" /></td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>OPEN/RETURN</td>
<td><img src="image9" alt="Diagram" /></td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>RETURN/UP</td>
<td><img src="image10" alt="Diagram" /></td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>OPEN/UP/RETURN</td>
<td><img src="image11" alt="Diagram" /></td>
<td>1000</td>
</tr>
</tbody>
</table>

**Table 3 Logic Structures**

Referring to Table 3, the attribute of an input contact, say, contact “1” at the beginning of a rung in the Ladder Diagram is OPEN and subsequent contacts, “2” and “3”, forming a Logical AND have no attributes. Starting Logical OR contacts such as “4” in a rung contains OPEN/RETURN attributes, middle OR contacts such as “5” have OPEN/RETURN/UP attribute and end contacts such as “6” have only UP attributes. The Ladder Diagram always has the rungs of ‘AND’ or ‘OR’ connected between the power rails. Each attribute in a rung must be checked for the power available in the logic circuit to analyze a Ladder Diagram.
[4] DEVELOPMENT OF ALGORITHM FOR PROCESSING ATTRIBUTES OF A LADDER DIAGRAM

The algorithm for various attributes of a Ladder diagram listed in Table 2 is developed using the Algorithmic State Machine (ASM) charts and is presented in Fig. 2. There are eight unique attributes used in the realization of ladder diagrams as was described in Section 2. The attributes of the ladder are read from the Code Memory in the form of 3-bit binary code. The functionality of each of these attributes is as follows.

In each attribute step, the ‘contact_status’ signal is checked. This signal shows the status of the input contact which is read from an I/O Image. If the value is “1”, then power available to that input contact is considered to be “1”, else it is taken as “0”. Thereafter, the control branches to state “0” to read next 16-bit User defined code from Code Memory. The stack write and stack read operations are performed at the location addressed by stack pointer. The ‘pwra’ signal holds the information of power availability to an input contact. The signals ‘pwrb’ and ‘pwrc’ holds the data of ‘pwra’ at various strategic points of the algorithm for different attributes. States in which various operations of the attributes are performed is illustrated in Fig. 2a and Fig. 2b.

As shown in Fig. 2, the “NIL” (or NO) attribute checks for the status of the input marked as ‘contact_status’ and returns to state “0” for processing the next contact. State “1” processes the “OPEN” attribute and stores the data of ‘pwra’ in stack1. The “RETURN” attribute performs stack write operation in stack2 and stack read operation from stack1. The “UP” attribute reads data from stack2. The data bits of ‘pwra’ and ‘pwrb’ are “OR”ed and the result is stored in ‘pwra’. The “OPEN/UP” attribute performs write operation in stack1 and stack read operation from stack2. Here also the data bits of ‘pwra’ and ‘pwrb’ are “OR”ed and the result is stored in ‘pwra’.

The “OPEN/RETURN” attribute stores the data bit value of ‘pwra’ in ‘pwrb’. The stack read operation from stack2 is performed in “UP/RETURN” attribute algorithm. For this attribute, “pwra’ and ‘pwrb’ are “OR”ed together and the result is stored in ‘pwrb’. Thereafter, the stack write operation is performed in stack2. For “OPEN/UP/RETURN” attribute, the ‘pwra’ value is assigned to ‘pwrc’. Subsequently, the ‘pwrc’ and ‘pwrb’ are “OR”ed together and the result is stored in ‘pwrb’.

![Algorithm Diagram](image-url)
Figure: 2. a. ASM Chart of the Attributes “0001” to “0010” (1-2)
[4.1] MEMORY DESIGN

There are two types of memories used in our design for realizing the logic of a Ladder Diagram. They are the Stack memory to store intermediate results and the Code memory to store the User Program. Input/Output Image is also designed to store the status of the input contacts and outputs. This section describes each of these memories in detail along with their ASM harts.

[4.1.1] STACK MEMORY DESIGN

The Stack memory comprises two numbers of stacks of size 4x1 bit each. For stack read, write, and clear operations, two stack pointers are defined which is of two bits each. The operation of the stack memory is described in terms of states and is illustrated in ASM chart shown in Fig. 3a and Fig. 3b. When the power on ‘reset_n’ signal is asserted, the ‘stack_state’ and ‘stack_operation’ signals will be reset. Also, by default, these two signals will be in “0” state. The write, read, and clear operations of stack1 are performed in state ‘1’, ‘2’ and ‘5’ respectively. Similarly the stack operations of stack2 are performed in state ‘3’, ‘4’ and ‘6’ respectively.

Figure: 2. b. ASM Chart of the Attributes “0011” and “1000” (3 - 8)
[4.1.2] CODE MEMORY DESIGN

The Code memory is used to store the User defined program and is of size 1024 x 16. The Memory Read operation is activated in state “0”. The 16-bit User defined code read from Code Memory is split into three signals in the format shown in Fig. 4 a. The ‘code’ represents the Normally-Open or Normally-Closed contact. The ‘attributes’ represents the type of the attribute defined to read the ladder diagram (See Table 2). The ‘io_address’ holds the address of the input/output of the I/O Image. The operation of the Code memory is described in terms of states and is illustrated in ASM chart shown in Fig. 4 b.

Two pointers ‘waddr’ and ‘raddr’ are used to point to the memory location during memory write and read operations respectively. When ‘reset_n’ signal is active, both address pointers ‘waddr’ and ‘raddr’ are initialized. After each memory write or read operation, these pointers are incremented automatically by “1” to point to the next memory location. The address bits used for 1K word memory locations are 10 bits and the data width is 16 bits. The memory write operation is performed in state ‘0’ and memory read operation is performed in state ‘1’.

[4.1.3] INPUT/OUTPUT/FLAG IMAGE

Image is of size 1024 x 1 bit and houses the images of 256 Inputs, 256 Outputs and 512 Flags. With power on reset, all the outputs of the Image are cleared. When the signal ‘IO_read’ is activated, memory read operation is performed to read the status of the input contact; else the output of the Ladder Diagram is stored in the output locations of the Image memory as depicted in Fig. 5.
Shobha S and Ramachandran S

Figure: 4. a. Format of the 16 bit User Defined Code

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes (bit 15) &amp; code(13 &amp; 14)</td>
<td>attributes</td>
<td>io_address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Selection of Memory operation

0

If write_mem = 1?

\[
\text{mem} \left[ \text{waddr} \right] \leftarrow \text{code} \text{in}
\]

1

waddr \leftarrow waddr + 1

TO state “0”

If read_code = 1?

\[
\text{code} \text{out} \leftarrow \text{mem} \left[ \text{raddr} \right]
\]

2

code \leftarrow \text{code out}[12:3]

\[
\text{io} \text{address} \leftarrow \text{code} \text{out}[9:0]
\]

raddr \leftarrow raddr + 1

execute \leftarrow 1

Default

TO state “0”

Figure: 4. b. ASM Chart of Code Memory

0

If reset_n = 0?

\[
\text{contact} \text{status} \leftarrow 0
\]

\[
\text{IO} \text{mem}[O \text{addr}] \leftarrow 0
\]

To state “0”

If IO_read = 1?

\[
\text{contact} \text{status} \leftarrow \text{IO} \text{mem}[io \text{address}]
\]

To state “0”

\[
\text{IO} \text{mem}[O \text{addr}] \leftarrow \text{coil}
\]

To state “0”

Figure: 5. ASM Chart of Input/Output Image
[4] SIMULATION RESULTS AND DISCUSSIONS

Algorithms for various attributes for solving a Ladder Logic were presented in Section 2. These attributes were coded in Verilog as per RTL coding guidelines. In order to verify their functionality, a test bench was written. The RTL design was simulated using ModelSim. An example Ladder Logic is used for demonstrating the working of the design and is shown in Fig. 6. As can be seen from the figure and Table 2, the attribute for contact ‘9’ is “OPEN/RETURN”. Similarly, the attributes of other contacts may be inferred. The output of the Ladder Diagram CR 316 indicates that it is the Control Relay.

The ‘reset_n’ signal is the power on reset signal. The ‘clk’ signal is the system clock which runs at 100 MHz frequency. The signal ‘read_code’ is used to read the 16 bit user defined code from code memory. During each memory read operation, the code that is read from the memory is assigned to the signal ‘code_out’. The signal ‘coil’ represents the output of the ladder diagram, CR316 in this case. The signal ‘pwra’ represents the power availability on the left side of the input contact “9” of the Ladder Diagram. Two stack memories ‘stack1’ and ‘stack2’ are designed to hold the intermediate results. In each memory read operation based on the signal ‘attribute’, the algorithm is executed. The signal ‘contact_status’ represents the functioning of the input contact. The signal ‘IO_read’ is used to read the status of the input contact and the output coil.

There are five paths to energize the output in a Ladder diagram shown in Fig. 6. Those paths are contacts “9” “1” and “2”, “3”, “4”, “1” and “5”, “6”, “1” and “5”, “7”, “8” and the reverse contact path given by “9”, “6”, “7”, “8”. In Fig. 7 all contacts referred by ‘io_address’ are cleared, hence signal ‘contact_status’ is ‘0’ for all contacts which indicates none of the contacts is functioning and hence the output coil is not energized. Subsequently the waveforms in Fig. 8 to Fig. 12 reveal the working of the five paths mentioned above and Fig. 13 shows the functioning of all the contacts. Fig. 8 depicts that the contacts “9” and “1” are activated. The ‘contact_status’ and ‘pwra’ are ‘1’ for these contacts. The contact “9” stores ‘pwra’ value in stack1 pointed by ‘stackpointer1’. As seen in figure at 325 ns the coil is energized after the ‘code’ gets the binary value “10”. The status of the coil is then stored at the address pointed by ‘O_addr’ signal in I/O image.

![Figure 6 Ladder Diagram Logic, an Example for Simulation](image-url)
The contacts “2”, “3”, “4” and “1” which is a path of the Ladder diagram shown in Fig. 6 are activated in Fig. 9. At time 145 ns the contact “2” is read from Code Memory as seen from figure. The algorithm defined for this attribute is executed before reading the next contact. The ‘contact_status’ and ‘pwra’ are ‘1’ for these contacts. After executing the algorithm of these contacts the coil is energized. Similarly in Fig. 10 contacts “5”, “6”, and “1” are activated. Coil is energized at time 535 ns. Also, the same status is there for the paths having contacts “5”, “7”, and “8” which is presented in Fig. 11 and the reverse path having “9”, “6”, “7” and “8” contacts which is depicted in Fig. 12.
Figure 9 Execution of Ladder Diagram in Fig. 6: “2”, “3”, “4” and “1” Input Contacts are activated.

Figure 10 Execution of Ladder Diagram in Fig. 6: “5”, “6” and “1” Input Contacts are activated.
Figure 11 Execution of Ladder Diagram in Fig. 6: “5”, “7” and “8” Input Contacts are activated

Figure 12 Execution of Ladder Diagram in Fig. 6: “9”, “6”, “7” and “8” Input Contacts are activated
The simulation result shown in Fig. 13 illustrates the functioning of all the contacts which energized the coil. The above results indicates that the defined algorithm for all attributes works based on the signals ‘contact_status’ and ‘pwra’. When power is available for all contacts in the path, their defined algorithm is executed and the coil is energized.

[5] CONCLUSIONS

New algorithm has been developed for various attributes for realizing logic of a Ladder diagram of a Programmable Controller. The design is realized using RTL Verilog. The user program is stored in the Code Memory. During memory read operation, each contact of the user ladder program is read from the Code Memory, decoded and executed. Stack Memory is used to store the intermediate data during the execution of the attributes. Image Memory is used to store the status of the input contacts as well as the outputs of the Ladder Diagram. Test bench has been developed in Verilog and the design has been successfully simulated using Modelsim. Numerous types of complex Ladder Diagrams have been tested. Currently, the design of Timers/Counters, Shift Registers, Drum Counters and arithmetic operations of Ladder Diagram based Programmable Controller is under progress.

REFERENCES


Conference on Electronic Measurement and Instruments ICEMI’09, Beijing, 16-19 August 2009, pp.4-778- 4-782.


