Dynamic Parse Approach of Ladder Diagram Based on Binary Logic Tree on the Soft PLC System

Guanci Yang, Jing Yang
Shaobo Li Guizhou University
China guanci_yang@163.com
627486628@qq.com lishaobo@gzu.edu.cn

Journal of Digital Information Management

ABSTRACT: *This paper mainly aimed to investigate the compiling problem of ladder diagrams (LDs) that coordinate with a soft-PLC system running on PCs or embedded intelligent terminals to accomplish corresponding control functions. The data structure to store the LDs was designed according to the series and the parallel logic relationship among LD instructions. Then the procedure to compile the LDs into a binary logic tree was detailed by adopting the principles of the binary tree, in which the child leaves represent the instructions, and the father nodes reflect their series or parallel relationship. Post-order traverse of the binary logic was used to parse the control logic, and logical expression was employed to proof the correctness of the transformation. The proposed method was compared with one algorithm. Statistical results on time overhead and interrupt response time showed that the proposed method performs better in most cases, indicating its superior capability to run the control logic. Finally, an example of the application in the central air conditioning control area is discussed as the case study to illustrate the proposed methodology.*

Subject Categories and Descriptors
F.4 [Mathematical Logic and Formal Languages]; Logic and constraint programming **B.6 [Logic Design]: I.2.5 [Programming Languages and Software]**

General Terms: Program Logic Controllers, Control models, Control logic, Ladder Diagrams, Soft-PLC systems

Keywords: Soft PLC system, Ladder diagrams, Binary tree, Energy conservation system

Received:18 May 2017, Revised 2 July 2017, Accepted 9 July 2017

1. Introduction

With their increasing sophistication, programmable logic controllers (PLCs) are finding novel applications on a large scale [21], and there are a lot of application of PLCs in market [2]. One of the kernel technologies of PLCs, which have applications in complex automation systems, is the design of the upper software. Software PLC (Soft PLC) is a type of control system basis for the structure of an embedded system or an industrial PC. Given the formulation and implementation of IEC61131-3 in the industrial control for PLC, a fully functional PLC process controller can be developed with software technology based on the IEC61131-3 protocol [1]. At present, Soft PLC technology is a hot topic because of its flexible development, open architecture, high cost-effectiveness, easy linking to networks, portability and available programming, among other properties [2, 3].Control models are commonly translated into ladder diagrams (LD) [4]. The motivation for figuring out the sequential control logic in LDs was to enable engineers, technicians and other users to develop software without the need for additional training, such as mastering a language such

as C/C++. The development, portability/reusability and maintenance of LDs simplified the similarity for familiar relay hardware systems. Considering that Energy conservation becomes a main theme nowadays in research [5], in previous works [6, 7], the focus was on development of soft PLC system for the energy-conserving control of the central air-conditioning. Thus, the graphic symbol was created and its functions were realised based on the QT Creator platform. Otherwise, we finished the visualisation design of the instructions and developed the user graphic programming interface (UGPI) for the graphic programming of the control logics. This study aimed to investigate the compiling problem of LDs running on PCs or embedded intelligent terminals to accomplish corresponding control functions. The data structure to store the LDs was designed according to the series and parallel logic relationship among LD elements. Then the procedure to compile the LDs into a binary logic tree was detailed by adopting the principle of the binary tree, in which the child leaves represent the instructions and the father nodes reflect their series or parallel logic relationship. Post-order traversal of the logic binary was used to parse the control logic, and logical expression was employed to prove the correctness of the transformation.

2. Related Works

Various products nowadays adopt soft PLC in applications ranging from simple machines to complete distributed plant-wide process controls [8]. An implemented distributed industrial automation based on soft PLC includes a control loop, which is composed of a remote controller, various sensors and a local execution unit. The sensor obtains the status of controlled plants and sends them to the remote controller through the network. The remote controller then calculates the controlled output and transmits them to the local unit, which updates the status of the controlled plant according to the controlled output. The software working on those nodes perform a set of tasks, such as receiving the status of plants, dealing with the control outputs and sending data to actuators. The control performance is related to the control algorithms and the scheduling of the shared network bandwidth resource. Ladder logic is widely used to program PLCs, where sequential control of a process or operation is required. LD language exposes easily understood visual connections between logical components. Usually, LD language demands are translated into text language, and then the compiler would make executable instruction codes of the device.

The logic control target on the soft PLC system can be achieved in two ways [9]: 1) simulation of LD based on an instruction list that translates the LD into an instruction list, which is then compiled into the objective code that can be recognised by the hardware; and 2) LD parsing, in which the system scans the LD routine so as to parse the control logic [10]. To develop a compiling system of soft PLC based on the instruction list, the design process of LD grammar was also investigated. This kind of system needs to define the function of each instruction, and its implementation is very complex [11].

Some studies [12, 13] phased the LD directly by recurrently scanning the rung to obtain the series and parallel relationship by judging the junction among the relays, contacts, instruction and rungs, which are characterised by low efficiency and high requirements of CPU and RAM. Tan et al. [14] used the idea of solving the maze problem to translate LD into instruction lists [15]. This method employed maze algorithm to identify the connected relation of the single node in the LD first, followed by the united serial or parallel relationship of each node, and then stored the serial or parallel connections of all nodes in binary trees. Finally, instruction lists were generated by scanning binary trees. This method determined that the maze algorithm could identifythe connected relation of LD nodesefficiently and correctly and guarantee the correctness of translating LD into instruction lists. However, tests results regarding this capability are limited.

Wannagat et al. [16] investigated multi-agent systems for PLC-based automation software using IEC 61131-3, and introduceda processing model to implement decentralised PLCs facilitating the dynamic reconfiguration of automation software in real time. Lai et al [22] designed an embedded soft PLC to help the developed intelligent control algorithm conveniently apply to actual controller, in which the Intelligent control algorithm can generate structuring text language and it can be loaded into PLC development software automatically. Schütz et al. [17] presentedthe ingenuity of a notion to develop and implement real-time-capable industrial automation software that improved the dependability of production automation systems by enhancing the implementation approach for PLCs. A type of methodology that converts LDs into a function block diagrams was presented for discrete control based on Fieldbus[18], which proposes an efficient way to implement discrete control based on intelligent field nodes. Later, asoft PLC system based on WorldFIP Fieldbus was created [19]. This system integrated the link activity scheduler and field-intelligent I/O modules and provided the functionof discrete control logic.Its UGPI provided users with an IDE for the configuration program. Discrete control logic was programmed in programming languages specified by IEC61131-3,and output data in the form of source file to the compiler. At the same time, the IDE could be employed to configure I/O points utilised by control logic. Its user configuration program compiler is to compile of users' configuration program and output TIC. Syntax check was executed before compilation. Field I/O points data configured in UGPI were compiled to data withtherequired format by the compiler.

3. Design of Data Structures for Ladder Diagram Storage

Ladder logic can be recognised as a rule-based language

rather than a procedural language. A 'rung' in the ladder means a rule. When implemented with relays and other electromechanical devices, the various rules 'execute' simultaneously and at realtime. When carried out in a PLC, the rules are typically executed sequentially by software in a continuous loop (scan). LDs are called 'ladder' diagrams because they resemble a ladder with two vertical rails (supply power) and as many 'rungs' (horizontal lines) as there are control circuits to represent. The LD is composed of rungs and instruction components. It consists of several rungs, and each rung contains some different elements.

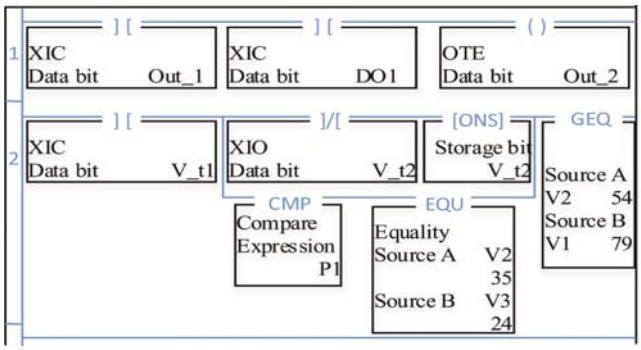


Figure 1. Illustration of ladder diagram

Figure 1 is a kind of LD thatincludes two rungs. The XIC instruction examines the data bit to determine if it is set and tests the data bit to determine if it is cleared. The ONS instruction enables or disables the remainder of the rung, depending on the status of the storage bit. When the rung is enabled and the storage bit is cleared, the ONS instruction enables the remaining fragment of the rung. When the rung is disabled or when the storage bit is set, the ONS instruction disables the remainder of the rung. The OTE instruction sets or clears the data bit. When the OTE instruction is enabled, the controller sets the data bit. When the OTE instruction is disabled, the controller cleans the data bit. The CMP instruction comparesthe arithmetic operations specified in the expression. The execution of a CMP instruction is slightly slower and uses more memory than the implementation of other comparison instructions. The advantage of the CMP instruction is that it allows you to enter complex expressions in one instruction. The EQU instruction tests whether Source A is equal to Source B. Strings are equal if their characters match. Moreover, ASCII characters are case sensitive, whichmeans upper case 'B' (42) is not equal to lower case 'b' (62). In the first rung, three instruction components exist, and the relationship between components is in series. The second rung contains six elements and a parallel circuit.

Given that the parallel and series relationship among instruction elements can be represented by a forward and backward relationship, a two-dimensional linked list was designed to store the LD (Figure 2). In such a list, one dimension is used to represent the relationship of rungs, which is a singlylinked list, and the other is employed to store the relationship of the instruction component, which is a doubly linked list. Each node contains the next-node link, a second link field pointing to the previous node in the sequence. The two links may be called 'forward('s') and 'backwards', or 'frontList' and 'nextList', in which the forward and backward instructions are stored respectively. When one instruction component is added into a rung, the doubly linked list should be updated by modifying the forward and backward relationship. If there are *m* rungs and the maximum number of instructions in the rungs is *n*, then the time complexity to find a particular instruction is *O*(*nm*). The dotted line shows the search path, which explores the rungs first, and then focuses on the doubly linked list.

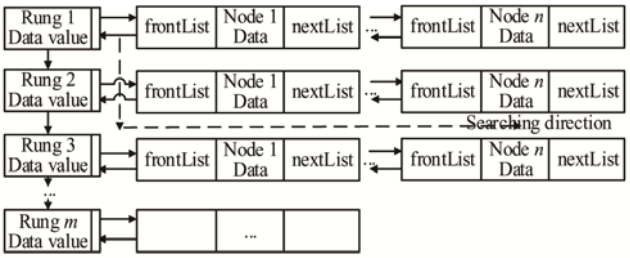


Figure 2. Illustration of the data structure for storing the LD

Class *SoftPLC*_graphicsItem is inherited from class QGraphicsItem providedby *QT*. The abstract data structure of a node can be defined by

```
struct BinTree
{int nodeType;
SoftPLC_graphicsItem * item;// QGraphicsItem
BinTree *lchild;
BinTree *rchild;
};
```

In addition, the general type definition of GT of Data is

```
typedef struct GeneralType
{string GT_Name; // variable name.
string GT_Type; // variable type of data.
string GT_Value;// variable value of data.
string GT_Desc;// variable description of data.
string GT_ID;// one unique ID to identify the variable.
}GT;
```

4. Dynamic Parse Algorithm of LD Based on the Binary Logic Tree

4.1 Proposed Algorithm Framework of DPA-LD-BLT

Compiling the ladder logic entails scanning the LD according to top-to-bottom and left-to-right scanning [10, 15].With the use of the designed storage structure, each instruction is stored in a double node, and the rung is an ordered doubly linked list. Thus, this type of list can be presented as abinary tree, and all the rungs,includingall the instructions,can be expressed as an ordered binary forest. The set of instructions can be obtained by post-orderingthe forest. According to the logic relationship among the instructions, if two instructions have a series connection, set anAND node as the parent node, and the enable output of the left child is the enable input of the

right child. If two instructions have aparallel connection, set an OR node as its parent node, and the logical OR result of those twoinstructions' enable output isthe enable input of the following instruction. Regarding promotion, the enable output of multi-parallel sub-rungs results in the enable input of the next LD. On the basis ofthis understanding, the dynamic parse algorithm of LDbased on the binary logic tree (DPA-LD-BLT) can be detailed as Figure 3.

```
Input: ladder diagram with n rungs, D ={R1,R2,...Ri,...,Rn}
Global variable: list AndItemList and OrItemList
        SoftPLC_graphicsItem* AndItem
        SoftPLC_graphicsItem* OrItem
Output: logic binary tree LBT-Tree
begin
01 BinTree* T[n]; int i: = 0;
02 for each rung Ri in D
03   if (rung Ri is not null) then
04       clear AndItemList and OrItemList,
05       for each item Ij in rung Ri
06            if (item Ij has no forward item) then list OrItem
             List append Ij
07       Sort OrItemList according the ordinate value of the
         items.
08       assign the front of OrItemList to p;
09       if (the count of list OrItemList is equal to 1) then
10                TreeNode: = CreateAndNode( p),
                  update AndItem and OrItem;
11       else TreeNode: = CreateOrNode(),
         update AndItem and OrItem;
12       while( AndItem or OrItem is not null )
13                if (AndItem is not null) then
14                   TreeNode: = CreateBinTree(TreeNode,
                    AndItem, TreeNode);
15                else if (OrItemList is not null) then
16                   OrItem: = the nearest backward item of
                    OrList;
17                   Sort OrItemList according the count of the
                    items' belongs.
18                   orNode: = CreateOrNode();
19                   TreeNode:= CreateAndTree(TreeNode,
                    orNode, TreeNode);
20       T[i]: = TreeNode;
21 Output the T;
```

Figure 3. Pseudo code of the proposed DPA-LD-BLT

In figure 3, *AndItemList* and *OrItemList* are used to store the contacts of parallel and series respectively, *AndItem* is employed to mark the item after the end of parallel branch, and *OrItem* is adopted to mark the item before the beginning of parallel branch. Function *CreateAndNode*, *CreateOrNode*, *CreateBinTree* and *CreateAndTree* are provided in the next sections.

4.2 Method to Create AND, OR Node, and Sub-Binary Tree

We designed two methodsto create *AND* and *OR* node, which are shown in Figure 4 and Figure 5, respectively.

In Figure 4(a), the input data **e* refers to the first node of series branch, and list *AndItemList* and *OrItemList* are used to save the contacts of parallel and series, respectively, and *AndItem* is used to mark the item after the end of parallel branch, and *OrItem* commemorates the item before the beginning of parallel branch. if *AndItemList*. count() >= 2, it means that there has more than two item and the new node has two children. After call this func-

```
Procedure: CreateAndNode(SoftPLC_graphicsItem *e)
begin
01 Define a variable: BinTree *andNode;
02 add item e to list AndItemList
03 while( e only has one backward item && the backward
item of e only has one forward item)
04    e : = the forward item of e;
05    add item e to list AndItemList;
06 if (e has more than one backward item && the nearest
   backward item of e only has one forward item)
07 then OrItem : = e, AndItem: = NULL;
08 else if (e only has one backward item && the nearest
backward item of e has more than one forward item)
09   then OrItem : = NULL, AndItem: = the backward item of e;
10   else if (item e has no forward item) then OrItem : = NULL,
AndItem: = NULL;
11 if (list AndItemList.count() > = 2 ) then
12    define two variables: BinTree *lchild,*rchild;
13   lchild->item: = the first item of AndItemList;
14   rchild->item: = the second item of AndItemList
15   Delete the first and the second item form AndItemList;
16   Create a new node by call function: andNode: =
      CreateAndTree(lchild, rchild, andNode);
17 else // new node has only one child.
18 andNode->item: = the first item of AndItemList;
19 Delete the first item from AndItemList;
20 return andNode;
```

(a)

```
Input: BinTree *btLChild , *btRchild , *andNode ;
Global variable: list AndItemList to store the contacts of
series;
Output: tree node
Procedure: CreateAndTree (BinTree *btLChild, BinTree
            *btRchild, BinTree *&andNode)
01       andNode->lchild: = btLChild
02       andNode ->rchild: = btRchild;
03       while(there is at least one item in list AndItemList)
04         if( AndItemList is not null ) then
05              Define node: BinTree *rNode;
06              rNode->item: = AndItemList, and assign left
                and right child of rNode are null;
07              Delete the front of AndItemList;
08              Call function to create a new node: andNode
                = CreateAndTree(andNode, rNode, andNode);
09       else break;
10 return andNode;
```

(b)

Figure 4. Pseudo code of creating an *AND* node

tion, it will return the parent node of the given series branch. Figure 4(b)is another function to create another kind of AND node with instruction leaves, in which *btLChild*, *btRchild* and *andNode* are the left sub-tree, right sub-tree and parent node, respectively.

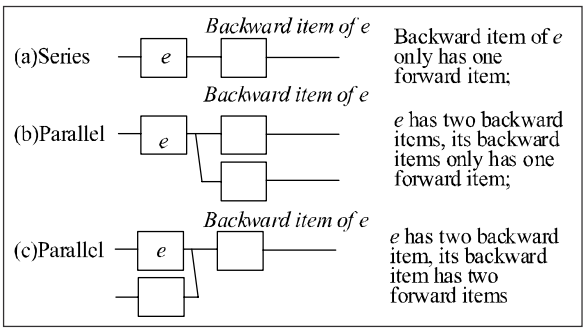


Figure 5. Illustration of the series and parallel relationships

The series or parallel relationships (Figure 5) are used in Figures 4. For each instruction, check the backwards of the instruction. If one instruction has more than one backwards, then create an AND node as the parent node of the nearest two items, and treat this AND node as one of the children of the next AND node until there are no other serial instructions. Finally, one sub-binary tree can be created (Figure 4). If the instruction follows a parallel branch, select two of the parallel branches and then create one OR node for them. Treat this OR node as one of the branches until there is no other parallel branch. Finally, one sub-binary tree can be created.

```
Procedure: CreateOrNode()
begin
01. Define nodes: BinTree *orNode,*lchild,*rchild;
02. Call function to create the left child:
    lchild = CreateAndNode( the first item of OrItemList );
03. Delete the first item from OrItemList;
04. Call function to create the right child:
    rchild = CreateAndNode( the first item of OrItemList );
05. Delete the first item from OrItemList;
06. Call function to create parent node:
    orNode = CreateOrTree(lchild, rchild, orNode);
07. return orNode;
```

Figure 6. Pseudo code of creating an *OR* node

Figure 6 is used to create an OR node without input parameters, which means that this type of sub-LD only has two branches, with each branch including only one instruction.

```
Input: BinTree *andNode
SoftPLC_graphicsItem *e
BinTree *&andTreeNode
Global variable: list AndItemList and OrItemList to
store the contacts of parallel and series, respectively
Output: root of a binary tree
Procedure: CreateBinTree(BinTree *andNode,
SoftPLC_graphicsItem *e, BinTree *&andTreeNode)
01  if(AndItemList.count() == 0)
02      add item e to list AndItemList
03      while( e only has one backward item && the
        backward item of e only has one forward item)
04          e := the forward item of e;
05          add item e to list AndItemList;
06      if( e has more than one backward item && the near-
        est backward item of e only has one forward item)
07      then OrItem : = e, AndItem: = NULL;
08      else if( e only has one backward item && the
        nearest backward item of e has more than one
        forward item)
09          then OrItem : = NULL, AndItem: = the
            backward item of e;
10          else if( item e has no forward item) then Or
            Item : = NULL, AndItem: = NULL;
11  Define nodes: BinTree *rchild;
12  rchild ->item: = the first item of AndItemList, and
    assign left and right child of rchild are null
13  Delete the first item of AndItemList;
14  andTreeNode->item = NULL, andTreeNode->lchild =
     andNode, andTreeNode->rchild = rchild;
15  while(there is at least one item in list AndItemList )
16      andTreeNode = CreateBinTree(andTreeNode,
        the front of AndItemList, andTreeNode);
17  return andTreeNode;
18  If( OrItem is not null) then
19          OrItemList := OrItem->nextList;
20          Sort OrItemList according the ordinate
            value of the items.
21          orNode = CreateOrNode();
22          andTreeNode: = CreateAndTree(andTreeNode,
            orNode, andTreeNode);
23  return andTreeNode;
```

Figure 7. Pseudo code of creating a sub-binary tree

Figure 7 shows how tocreate a sub-binary tree, which includes an OR node and an AND node. Figure 8can produce an OR node with a given left child and right child.

```
Input: BinTree *btLChild //left sub-tree
BinTree *btRchild //rith sub-tree
BinTree *&andNode // parent node
Global variable: list OrItemList to store the contacts
of series; Output: tree node
Procedure: CreateOrTree(BinTree *btLChild, BinTree
*btRchild, BinTree *&orNode)
01 orNode->lchild: = btLChild
02 orNode->rchild: = btRchild;
03 while(there is at least one item in list OrItemList)
04 Create node rNode by call function:
rNode:= CreateAndNode( the front of OrItemList);
05 Delete the front of OrItemList;
06 Call function to create a new node:
orNode = CreateOrTree(orNode,rNode,orNode);
07 return orNode;
```

Figure 8. Pseudo code of creating an OR node with a given left child and right child

4.3 Correctness Proof of the DPA-LD-BLT

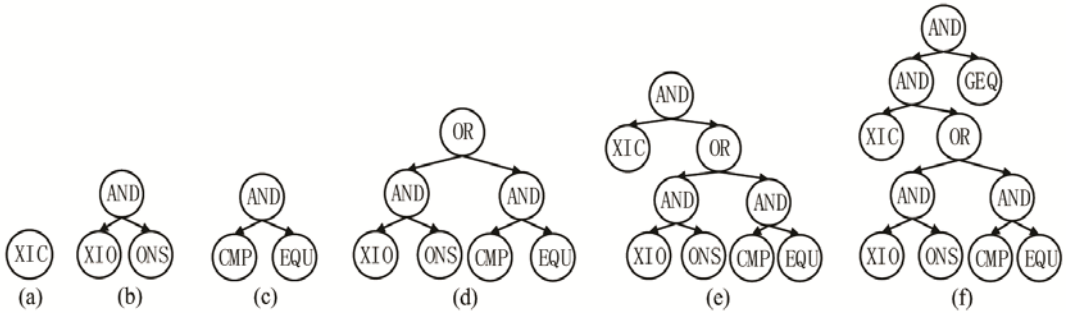


Figure 9. Illustration of the LD translated into a binary logic tree

According to the proposed DPA-LD-BLTalgorithm, the second rung in Figure 1 can be expressed as Figure 9. First, create a leaf node for the XIC instruction [Figure 9(a)]by calling the function shown in Figure 4. One of the parallel branches can be described by Figure 9(b), and theother parallel branch can be demonstrated by Figure 9(c). Given that these two branches are parallel merging,the twosub-treescan obtain the newsub-tree [Figure 9(d)]. This new sub-tree and the XIC node are serial, and they can be grown into Figure 9(e). Finally, the sub-tree shown in Figure 9(a) and the GEQ node are also serial, and they can be represented as Figure 9(f), which represents the total rung.

Before the running of ladder logics, set the input of rung to trigger the logic (Figure 10). The logic output of the instruction is the logic input of the next instructions. Figure 11 illustratesthe output and input of the rung in a binary tree.

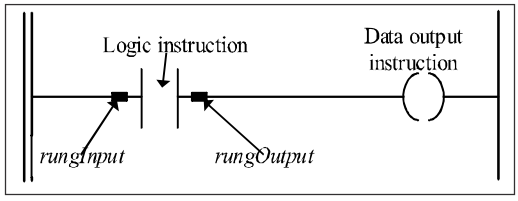


Figure 10. Illustration of the rung trigger condition

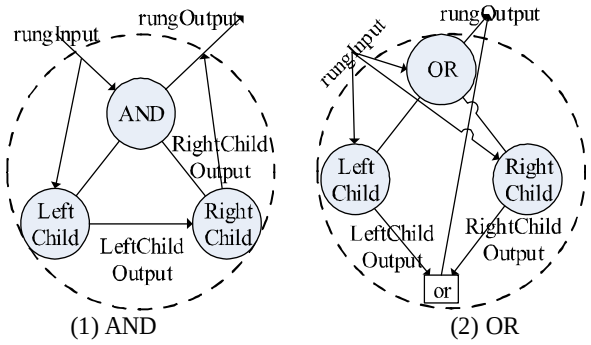
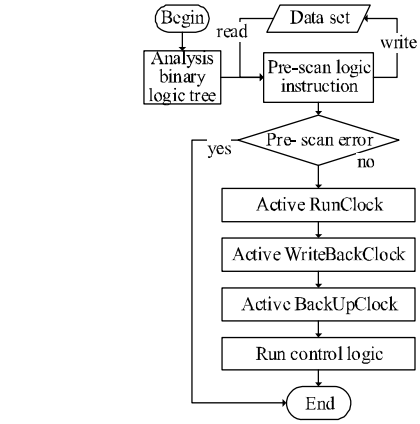
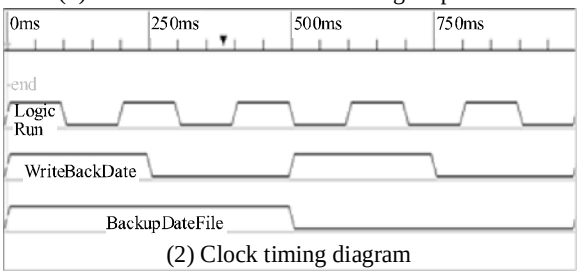


Figure 11. Logic relationship of the binary logic tree

The AND structure indicates the serial relationship, and the *rungInput* is the input of the left sub-tree/child. After the execution of the left sub-tree/child, its output is the input of the right sub-tree/child, which then triggers this part of logic. The output of the right child/sub-tree is the finaloutput of the AND structure. The OR structure represents the parallel relationship; the left child/sub-tree will be run first, and the following operation is the right child/sub-tree. The OR operation between the output of the left child/sub-tree and the right child/sub-tree is the final output of the control logic.



(1) External control flows of the logic operation



(2) Clock timing diagram

Figure 12. Clock timing diagram and external control flow of the logic operation

Figure 12 shows the clock timing diagram and external control flow of the logic operation. As a part of the soft PLC system, the system needs to pre-scan the logic instruction before the functioning of the binary logic tree. The run clock controls the overall system, the write back clock responds to the control of data update and the backup clock tells the system the right period toback up the current data. The soft PLC system will be run according to the given clock period (usually 200 ms).

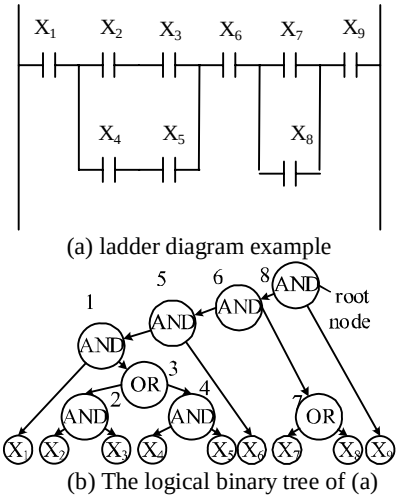


Figure 13. Typical ladder diagram and its logical binary tree

Figure 13 is a typical LD that has parallel and serial branches and nine relay elements. Figure 13(b) is the logical binary tree of (a) by employing the proposed approach in the previous section. If the rung input is TURE, then the control logical result Y can be expressed by

$$Y = X_1 \&\& ((X_2 \&\& X_3) \| (X_4 \&\& X_5)) \&\& X_6 \&\& (X_7 \| X_8) \&\& X_9.$$

The demonstration of correctness based on the typical ladder and the logical binary tree generated by the proposed algorithm is given following the logical expression below:

(1) According to the principle of in-order traversal of the binary tree, the leaf of X_1 is the first node that will be executed, and that means the X_1 is the first executions.

(2) The parent of X_1 is AND node 1, and the result of X_1 is the input of the right sub-tree. In-order traversal of the right sub-tree and the output of the left sub-tree of OR node 3 is $(X_2 \&\& X_3)$, and its right sub-tree results in $(X_4 \&\& X_5)$. Therefore, the final results of OR node three can be represented by $((X_2 \&\& X_3) \| (X_4 \&\& X_5))$.

(3) AND node 1 is the parent of node X_1 and OR node 3.According to the illustration of the AND structure shown in Figure 11, the running result of this sub-tree is $X_1 \&\& ((X_2 \&\& X_3) \| (X_4 \&\& X_5))$.

(4) Similarly, after the running of X_6, the logical tree can be represented by $X_1 \&\& ((X_2 \&\& X_3) \| (X_4 \&\& X_5)) \&\& X_6$.

(5) X_7 and X_8 share the same OR node 7, and the execution result is $(X_7 \| X_8)$.

(6) AND node 6 is the parent of OR node 7 and AND node 5; its results are $X_1 \&\& ((X_2 \&\& X_3) \| (X_4 \&\& X_5)) \&\& X_6 \&\& (X_7 \| X_8)$.

(7) The root node is an AND node, and the final result of the total logical tree can be represented as $X_1 \&\& ((X_2 \&\& X_3) \| (X_4 \&\& X_5)) \&\& X_6 \&\& (X_7 \| X_8) \&\& X_9$.

In conclusion, the results reported by the proposed algorithm are the same as those of logical reasoning, thereby proving the correctness of the proposed algorithm.

5. Experiments

5.1 Test Platform and Problem

We implemented the proposed data structure and the proposed DPA-LD-BLTalgorithm using C++ language intheQT platform, and integrated it with the soft PLC system,which has the following functions:1) Object configured function: to add the objects'entity, communicate with hardware, get the keys and so on. 2) Hardware configured function: to configure modules, write the key and point mappings for the module, configure ports and so on. 3) Control logicand strategies programming functions: to complete the system's controlling logicandcomplete the controlling of controlled equipment. 4) Data analysis and so on. The experiment mentioned lately is based on this system. One controller (Figure 14) was developed by our research group. The soft PLC, one oscilloscope and one PC were installed and employed as auxiliary equipment to test the performance of the proposed algorithm. Black box testing and load testing were also carried out.

We designed an LD to control six digital input units (DIUs), in which each DIU has four DO ports. The six DIUs are mounted to the used controller. When the first DO port of the first DIO module is switched on, it will return a value, which will be thetrigger input of the secondDO port, and the result of the second DO port will be the trigger input of the next port, and so on. Finally, after the 24 ports are switched on, they are all turned off simultaneously, over and over again. The system scan cycle is setto 100 ms, and the time consumption of the communication is 20 ms. The used LD includes bit manipulation, timer and counter, compare instruction, calculator and arithmetic instruction, transfer and logical instructions, program controller, loop and stop instructions and the particular fuzzy PID.

Figure 14. Controller used for the test

5.2 Compared Methods

To evaluate the performance improvement of our DPA-LD-

BLT, we compared our method with one algorithm proposed in Ref. [15], entitled 'The Application of Maze Algorithm in Translating Ladder Diagram into Instruction Lists of Programmable Logical Controller (MTLD). MTLD employed the maze algorithm to identify the connected relation of the single node in the LD first, followed by the united serial or parallel relationship of each node, and then stored the serial or parallel connections of allnodes in binary trees. Finally, instruction lists were generated by scanning binary trees. This method declared that maze algorithm can identify the connected relation of LD node efficiently and correctly and guarantee the correctness of translating LD intoinstruction lists. However, its tests results are limited. MTLD using C++ language was also implemented in the QT platform and treated as a part of the mentioned system.

5.3 Experimental Results Analysis

Figure 15presents the monitoring data of executing state of the DIU ports by using the oscilloscope, which indicates the control logic is correct. The time difference of state transition among ports of the DIO module when the scan cycle is 100 ms are shown in Table 1, in which T_1, T_2, T_3 and T_4 are the time differences of DO_1 to DO_2, DO_2 to DO_3, DO_3 to DO_4 and DO_4 to DO_1, respectively. According to the data, all the time differencesreported by DPA-LD-BLTbelong to [95ms, 105ms], both of which are smaller thantheMTLD of [96ms, 106ms].Figure 16is the box plot based on 10 independent trials. The chart belonging to DPA-LD-BLTin the box plot is located in the lower place with the smaller coverage area, which indicates that DSPEA outperforms the compared algorithm in terms of time overhead.

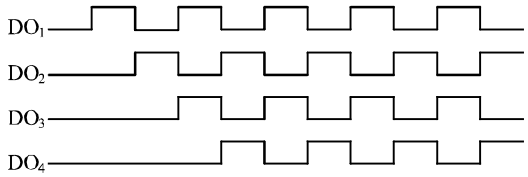


Figure 15. Executing state of the DIU ports

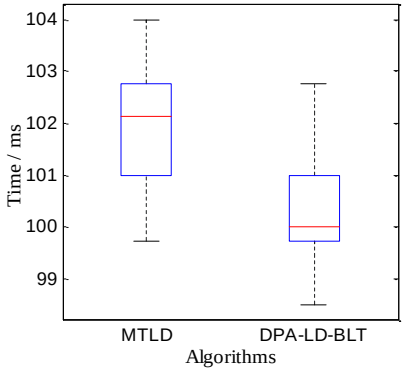


Figure 16. Boxplot of the time difference of state transition among ports of DIO module based on 10 independent trials

Considering the robustness of the system, the interrupt response time was checked when the scanc ycle is 20ms. The system completed the given control task steadily. Table 2 is the statistical interrupt response time.The maximum interrupt response time reported from the systemusing the proposed methodis 23ms, and the minimum time is 20.5ms. The time consumption of the communication is 20ms, and so the real-time use of the control logic is less than 3ms. By contrast, the system employing the MTLD reported maximum and minimum interrupt response times of 24.5 and 20.75ms respectively. According to the above statistical results, the proposed method performs better on time overhead and interrupt response time in most cases compared with the compared algorithms, indicating that DPA-LD-BLT exhibits superior capability in running the control logic.

Algorithms	Maximum time/ms	Minimum time/ms
DPA-LD-BLT	23	20.5
MTLD	24.5	20.75

Table 2. Interrupt response time reported from the system with different algorithms

No	DPA-LD-BLT				Average Time	MTLD				Average time
	T_1/ms	T_2/ms	T_3/ms	T_4/ms		T_1/ms	T_2/ms	T_3/ms	T_4/ms	
1	98	104	100	97	99.75	101	102	98	103	101
2	101	100	99	104	101	103	105	99	98	101.25
3	103	97	101	100	100.25	105	103	105	103	104
4	101	101	98	99	99.75	103	99	104	104	102.5
5	100	104	102	105	102.75	105	99	104	104	103
6	99	95	100	103	99.25	100	98	98	103	99.75
7	97	97	99	101	98.5	104	103	99	105	102.75
8	100	103	97	99	99.75	106	98	98	106	102
9	98	102	99	103	100.5	99	96	105	100	100
10	101	102	103	100	101.5	106	101	99	103	102.25

Table 1. Time difference of state transition among ports of the DIO module

6. Application Case

We used the soft PLC system to control one ACS transducer, and a simulation and debugging platform of central air conditioning system, embedded controller and efficient frequency control cabinet MSC-400 were employed, as shown in Figure 17.

(a)

(b)

Figure 17. Equipment used in the case study: (a) simulation and debugging platform of central air conditioning system, (b) efficient frequency control cabinet MSC-400, was which developed by Guizhou Huitong Huacheng CO., LTD

The simulation and debugging platform of the central air conditioning system is used to simulate the status data and load characteristic of the controlled systems/terminals/devices, as well as the required environmental parameter sampling, including delivery and return air temperature, concentrations of carbon dioxide, temperature and humidity of monitoring area. The embedded controller running the developed soft PLC system used as a configuration software is employed to program the control logic, and it will provide the optimal control parameters for the efficient frequency control cabinet MSC-400, which is obtained by running the optimal control algorithm according to the simulated data submitted by the simulation and debugging platform of central air conditioning system. The efficient frequency control cabinet MSC-400 is adopted to control the controlled systems/terminals/devices and feedback them dynamic information for the simulation and debugging platform of central air conditioning systems.

To check the performance, the scan cycles of the system are set to 300, 500 and 1000 ms, and then the system response times were recorded. A total of 12 independent trials were completed. Table 3 lists the system response times reported from the system with different scan cycles. According to Table 3, all the average times are less than 300 ms, which satisfy the requirements of an energy-saving control system [20]. The results show that the dynamic parse approach is capable of achieving the goal of control logic efficiently.

Scan cycle/ms	300	500	1000
System Response Time	298	326	226
	235	255	302
	302	320	305
	301	317	230
	334	273	239
	303	286	340
	309	280	360
	332	220	360
	298	288	320
	280	250	334
Average Time	299.2	281.5	292.6

Table 3. System response time reported from the system with different scan cycles in 12 independent trials

7. Conclusion

Soft PLC technology has become a hot topic because offlexible development, open architecture, high cost-effectiveness,easy linking to networks, portability and convenient programming. Nowadays, soft PLC technology is widely used in industry control systems. Our research focuseson the energy-saving control system and develops soft PLC systems to improve the efficiency of the energy-saving products. In this paper, we concentrate on the programming, compiling and execution of LD. Atype of dynamic parse algorithm of LD based on the logical binary tree is also investigated, which simplifies the instruction list-based method and has small time consumptions compared to the interpreter who translates a program as it is read, which indicates the algorithm has good application prospects. Experiments will also be conducted with more complex control logic and involving more complicated applications. Additionally, this work is part of our on-going research that explores robot control technologies in human–robot interaction (HRI). We aim to develop more natural, human-centred, proactive HRI mechanisms. We will investigate on how to use soft PLC technology to improve the robot control accuracy. Otherwise, we will focus on the transformations of intermediate form into machine code to improve system efficiency.

Acknowledgments

The authors would like to thank the editor and the

anonymous reviewersfor their constructive comments and suggestions to improve the quality of the paper.

References

[1] Hoffman A. J., Basson A. H. (2015). IEC 61131-3-based holonic control of a reconfigurable manufacturing subsystem. *International Journal of Computer Integrated Manufacturing,* 29 (5) 520-534.

[2] Alphonsus, E. R., Abdullah, M. O. (2016). A review on the applications of programmable logic controllers (plcs). *Renewable & Sustainable Energy Reviews* 601185-1205.

[3] Negahban A., Smith J.S. (2014). Simulation for manufacturing system design and operation: Literature review and analysis. *Journal of Manufacturing Systems* 33 (2) 241-261.

[4] Luo J., Ni H., Zhou M. (2015). Control program design for automated guided vehicle systems via Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45 (1) 44-55.

[5] Ababneh. A. (2011). Energy Conservation Using a Double-effect Absorption Cycle Driven by Solar Energy and Fossil Fuel, *Jordan Journal of Mechanical and Industrial Engineering* 5 (3) 213-219.

[6] Yang, G., Yang, J. (2017) ExecutingStrategy and Visualization Design for Instruction of Soft PLC System, *Journal of Residuals Science & Technology* 14 (3) 317-326.

[7] Dai, Z., Yang, G., Li, S. (2012). Analysis and Implement Graphical of Soft PLC Logical Control Elements. *Modular Machine Tool & Automatic Manufacturing Technique*, (10) 74-76.

[8] Niu, Y., Cao, L., Wu, X. (2012). Optimal co-design of control and scheduling for distributed industrial automation based on embedded soft PLC. *Advances in Information Sciences and Service Sciences* 4 (13) 135-143.

[9] Al, Y. I (2012). Design and Implement of a Programmable Logic Controller (PLC) for Classical Control Laboratory. *Intelligent Control and Automation,* 3 (1) 44-49.

[10] Wang, X., Zhou, F., Sun, S. (2008). Development and implementation of Soft PLC compiling system. *Journal of Beijing University of Technology*, 34 (11) 1139-1144.

[11] Huang, J. (2014). Engineering machinery research and realization of the embedded soft PLC programming environment. *Central South University* Changsha.

[12] Qin, Y., Dai, W., Yang, J. (2015). Design and experiment of operational control system for mineral grinding process based on soft PLC technology. *Journal of Northeastern University,* 36 (3) 309-313.

[13] Li, J. (2015). Research on Soft PLC in Embedded CNC System. *South China University of Technology* Guanzhou.

[14] Yadav, S., Verma, K. K., Mahanta, S. (2012). The Maze problem solved by Micro mouse. *International Journal of Engineering and Advanced Technology,* 2 (4) 157-162.

[15] Tan, A., Ju, C. (2011). The Application of Maze algorithm in Translating Ladder Diagram into Instruction Lists of Programmable Logical Controller *Procedia Engineering,* 15, 264-268.

[16] Wannagat, A., Vogel-Heuser, B. (2008). Increasing flexibility and availability of manufacturing systems-dynamic reconfiguration of automation software at runtime on sensor faults. *IFAC Proceedings,* 41(3) 278-283.

[17] Schütz, D., Wannagat, A., Legat, C. (2013). Development of PLC-based software for increasing the dependability of production automation systems. *IEEE Transactions on Industrial Informatics,* 9 (4) 2397-2406.

[18] Liang, G. (2008). A kind of function block application model for distributed intelligent control network based on WorldFIP. *In:* Proceedings of Chinese Control and Decision Conference Liang, Geng. A kind of function block application model for distributed intelligent control network based on WOrldFIP. Chinese Control and Decision Conference 1400-1404. IEEE.

[19] Liang, G., Li, Z., Li, W. (2012). On an LAS-integrated soft PLC system based on WorldFIP fieldbus, *ISA Transactions,* 51 (1) 170-180.

[20] Harris, T. J., Seppala, C. T., Desborough, L. D. (1999). A review of performance monitoring and assessment techniques for univariate and multivariate control systems, *Journal of Process Control,* 9 (1) 1-17.

[21] Rösch, S., Vogel-Heuser, B. (2017). A light-weight fault injection approach to test automated production system plc software in industrial practice. *Control Engineering Practice*, 58, 12-23.

[22] Lai., B, Li, Z., Xiong, J. 2017) The design of soft PLC controller which intelligent algorithms can be transplanted into conveniently. *Microcomputer & Its Applications*, 5. 1-3.