Int J Adv Manuf Technol (2010) 49:615–626
DOI 10.1007/s00170-009-2405-0

ORIGINAL ARTICLE

A study on the generation of silicon-based hardware Plc by means of the direct conversion of the ladder diagram to circuit design language

Daoshan Du · Xiaodong Xu · Kazuo Yamazaki

Received: 29 September 2008 / Accepted: 26 October 2009 / Published online: 18 November 2009

Abstract Programmable logic controller (PLC) is one of the most important components in today's manufacturing. Its performance-based microprocessor and software have been a bottleneck for improving its efficiency. To enhance the PLC performance and flexibility, a new PLC design based on field programmable gate array (FPGA) has been a hot topic because of its parallel execution mechanism and reconfigurable hardware structure. From practical viewpoint, in this paper, the authors propose an approach to implement the existing ladder diagram (LD) inside FPGA making full use of the advantage of FPGA device. The essential of this research includes two issues: (a) analyze the LD program and organize it with sequential and parallel structure and (b) implement the sequential and parallel structure of the LD program with hardware description language inside FPGA. To the first work, the condensed simultaneity graph theory is applied to optimize the LD program with sequential and parallel structure. To the second work, Boolean equations are taken as the bridge to convert the optimized LD program to the hardware description language program. Finite state machine is used to generate sensitive signals to guarantee that the performance of the converted very high-speed integrated circuit hardware description language design is the same as the original ladder diagram. A case study is practiced to verify the proposed approach in this paper.

Keywords Programmable logic controller · Ladder diagram · Very high-speed circuit design hard description language · Field programmable gate array · Condensed simultaneity graph · Finite state machine

D. Du (✉) · X. Xu · K. Yamazaki
IMS-Mechatronics Laboratory,
Department of Mechanical and Aeronautic Engineering,
University of California,
Davis, CA 95616, USA
e-mail: dudaoshan@hotmail.com

1 Introduction

Programmable logic controllers (PLCs) have been widely used for the logic control in the manufacturing system [1]. In industrial application, ladder diagram is the most popular programming language for PLC development. Traditionally, PLC includes a microprocessor, and the ladder diagram is sequentially executed inside this PLC microprocessor in a cyclic scan period. Based on this solution, PLC performance is limited by the cyclic scan period, which depends on the program length and the microprocessor's processing speed.

In order to overcome these drawbacks with the programmable hardware solution, field programmable gate array (FPGA)-based PLC has been focused on by many researchers with its reconfigurable hardware structure and parallel execution advantage. Miyazawa [2] and Ikeshita et al. [3] developed a very rough manner in 1999 to convert the graphic ladder diagram into program description of very high-speed integrated circuit hardware description language (VHDL). Chen and Patyra [4] designed a VHDL model of the whole system directly from the original system requirements to build a controller. Abdel-Hamid et al. [5] and Kuusilinna et al. [6] developed an algorithm to convert finite state machine (FSM) into VHDL. Adamski, M. et al.

[7–10] did an effective work in choosing Petri net model as a substitute of ladder diagram in manufacturing control. These studies show that reconfigurable hardware has advantages of simplicity of programming, size, and cost, while parallel execution of FPGA can improve the PLC performance dramatically.

In FPGA design, parallel execution exists not only in one combinational logic operation but also in the multiple combinational logic operations. To the first case, in one combinational logic operation, there is only one output. But, all of the combinational logics which influence this output have been designed in a flat way as the existing circuit. The operation can be executed at the electrical speed. To the second case, in the multiple combinational logic operations, all of these combinational logic operations that influence every output have been designed in a flat way as the existing circuits. So they can occur in a parallel way. For the first case, it is very easy to be realized in VHDL. In this paper, the parallel execution to be discussed here specially means the second case.

In addition, the current research stays at a very rough stage by only introducing case-based conversion from PLC's description language to VHDL. Also, most methods aim at obtaining the hardware description languages (HDL) or the register transistor level (RTL) architecture of FPGA from the original system requirement. As the majority of PLC programs in the manufacturing system are written by ladder diagram, it is essential to make use of the existing ladder diagram of current PLC system to conduct new PLC design.

It is in a different way to implement PLC ladder diagram between inside microprocessor and inside FPGA. In microprocessor, the PLC instruction is sequentially executed line by line. Otherwise, in FPGA, all of probabilities of the PLC instructions have been considered and designed with circuit, which can be activated by sensitive signals. In order to implement PLC ladder diagram inside FPGA, the sequential operations in the original ladder diagram should be kept. At the same time, the capability of parallel execution can be used to get more efficient performance. This paper will discuss how to implement the PLC ladder diagram in the sequential execution way and the parallel execution way inside FPGA. Programming language for FPGA is VHDL.

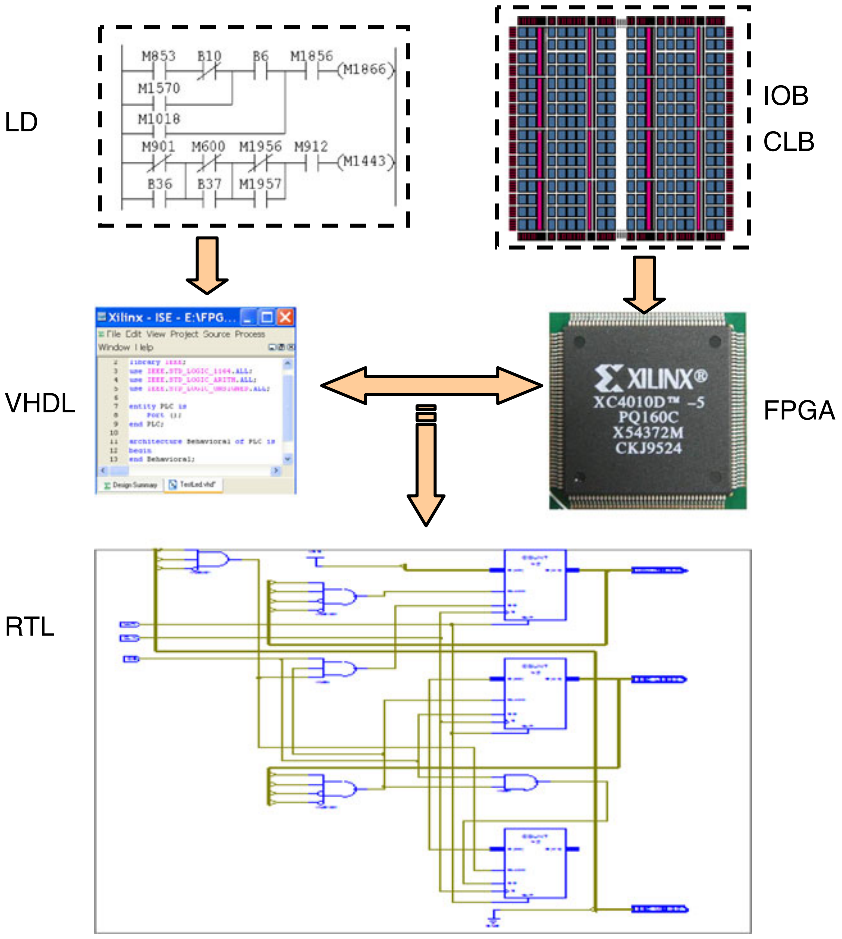
The left portion of this paper is organized as follows. Section 2 briefly introduces the FPGA-based PLC design, the research feasibility, and overall approach. Section 3 explains how to optimize the original ladder diagram with the sequential structure and parallel structure. The method to convert the optimized ladder diagram into VHDL is discussed in Section 4. A case study is shown to verify the proposed approach in Section 5. Finally, Section 6 concludes the paper with a brief description of the ongoing work and future research.

2 FPGA-based PLC design

FPGA technology has great advantages to conduct new PLC design by comparison with the traditional software-based PLC solution. FPGA-based PLC scheme can improve PLC performance, reduce manufacturing cost, and enforce flexibility of the logic control of the manufacturing system. In order to approach these goals, the essential point is to convert the PLC program into gate-level digital circuit expressions so that the same control logic of PLC represented in its program can be exactly reflected in the FPGA-based solution. Since the internal structure of FPGA is reconfigurable with input/output block and configurable logic block, the required circuit can be built as long as a ladder diagram is converted into RTL architecture and downloaded to the FPGA chip. Such an implementation will perform the same functions as the original PLC ladder diagram, but not in the traditional sequential cyclic scan manner. The new solution can respond to the input signals with parallel execution at hardware processing speed, which dramatically improve the PLC speed. Moreover, it can be reconfigured any times as a new PLC program is converted and downloaded.

As for the RTL architecture of FPGA implementation, it is a low-level description of internal logic circuit of FPGA. Several commercial software tools are available for FPGA development. The typical one is Xilinx ISE, which can establish RTL architecture by synthesizing a high-level program such as VHDL. Figure 1 shows the approach to implement a FPGA-based PLC from the ladder diagram. There are two steps to construct FPGA-based PLC: (a) convert ladder diagram into VHDL program with sequential and parallel design and (b) synthesize the VHDL design and implement it to FPGA device. With the special reconfigurable structure of the FPGA device, RTL architecture can be configured inside FPGA, which can execute the functions of the original ladder diagram inside FPGA with more efficiency and many other advantages such as capability of reconfiguration, low cost, etc.

Usually, in the traditional PLC system, there are three obvious characteristics: (a) The PLC system is based on the microprocessor, which works as the central processing unit (CPU) to sequentially process the instructions which comprised the PLC program; (b) The PLC program mostly works as the software and is compiled as object code, which is executed by the microprocessor; and (c) The PLC works with a cyclic scan. During one period, the microprocessor finishes once scanning of the whole PLC program. In this kind of working way due to the limit of microprocessor, all of these operations are exactly executed in a sequential way inside the CPU. In fact, PLC system is a discrete event control system; some events must occur after other events while some events can occur in parallel. If the



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PLC is
    Port ();
end PLC;

architecture Behavioral of PLC is
begin
end Behavioral;
```

Fig. 1 Implementation of FPGA-based PLC from ladder diagram

system is designed not only with the sequential structure as the traditional PLC program but also with parallel structure, it must be operated efficiently, and the performance can be dramatically improved.

With the programmable hardware design technology such as PLD, FPGA, and complex programmable logic device, it becomes possible to design the discrete event control system based the hardware platform. FPGA-based PLC has been generally discussed by many previous researchers [2–13]. From practical view point, it is a very valuable solution to implement FPGA-based PLC by converting ladder diagram, which is the most popular programming language, to VHDL program, which is also the most popular hardware description language for FPGA design. The ladder diagram is organized in a sequential way, while the FPGA design can be organized in both the sequential and parallel way. In order to efficiently implement the original ladder diagram inside FPGA, it is necessary to analyze the ladder diagram and reorganize it with both the sequential and parallel structure. Therefore, how to convert a sequential ladder diagram into VHDL program with sequential and parallel design is the key of this research.

3 Optimization of ladder diagram program with sequential and parallel structure

In this paper, the condensed simultaneity graph (CSG) theory [14, 15] is used to realize the reorganization of the ladder diagram in both sequential and parallel way.

3.1 The condensed simultaneity graph theory

As a discrete event control system, the output of every rung in the ladder diagram can be described as a state variable, which is related to the status of one event switch having two possible values, “0” or “1.” In ladder diagram, if the rung output depends upon the output of a previous rung or rungs, it is called dependent state. On the other hand, if the rung output is independent from the outputs of other rung or rungs and can occur at the same time with others in the ladder diagram, it is called independent state. Based on the CSG theory, two kinds of graph, the simultaneity graph and the dependency graph, can be used to describe those independent states and dependent states.

Here, a simple example of ladder diagram is shown as follows (Fig. 2), which will be used to explain the

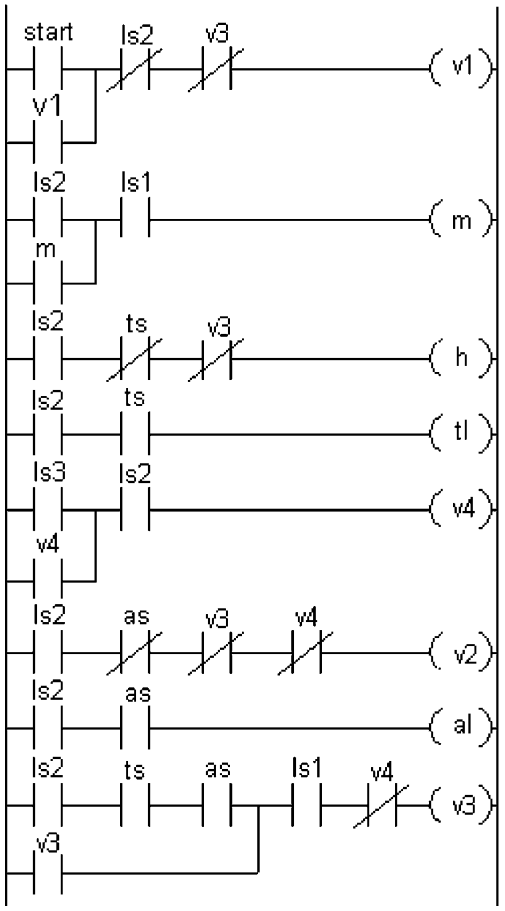


Fig. 2 An example of ladder diagram

condensed simultaneity graph theory. The system has eight state variables: v1, m, h, tl, v4, v2, al, and v3 and six inputs: start, ls1, ls2, ls3, ts, and as.

3.1.1 The condensed simultaneity graph

In order to describe the independent states and the dependent states in the graph, two elements used are as follows: a node is used to represent one state, which is on behalf of one output of rung in the ladder diagram; an undirected or directed line is used to indicate that the relationship of the two states on the both ends of the line is independent or dependent.

(1) The simultaneity graph (SG)

The first graph is the simultaneity graph (Fig. 3a). This graph uses undirected line to connect two states, which is contained if these two states can be on concurrently. This graph has a node for each rung output and an edge connecting the nodes corresponding to rung outputs that can be true simultaneously at the completion of one ladder diagram scan. The SG contains the sequential information of the process statements in VHDL program. The dependency information is very important to keep the necessary sequential operations in the original ladder diagram.

If the graph is simple and there are not many states and edges, the adjacent matrix data structure is used to store the graph, which is easier to be processed. Or else, in order to save memory, adjacent list date structure is a better way to store the graph, which is a general method. Adjacent list is taken to describe the SG graph in this paper.

(2) The dependency graph (DG)

The second graph is the dependency graph (Fig. 3b). This graph uses directed line to connect two states if one state depends on the output of previous rung or rungs at the completion of one ladder diagram scan. The DG has a node for each state variable in the ladder diagram. Since the DG is a directed graph, each edge is represented by a directed pair<u; v>; u is the tail and v the head of the edge and the edge<u; v>implies that rung v depends on the output of rung u. The DG contains the concurrency information of the process statements in the VHDL model. The concurrency information is very important to implement concurrent operations inside FPGA for high efficient performance.

(3) The condensed simultaneity graph (DSG)

In order to integrate all of the concurrency information and the dependency information into one graph, the condensed simultaneity graph is introduced (Fig. 3c); an undirected graph and created by condensing the nodes of the simultaneity graph that are connected in the dependency graph into one node. The node interconnections are the same as in the original simultaneity graph. In this graph, more than one rung output can be associated with a node, and it contains information on the dependency in the SG and the concurrency in the DG.

(4) Implementation of the condensed simultaneity graph

According to the theory described in from (a) to (c), the CSG of the example can be built. The steps are as follows:

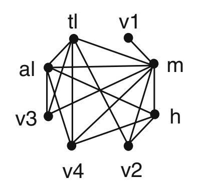
Step 1: Input the ladder diagram with instruction list expression, get through all of the rungs, count the rungs, and construct the list arrays for SG, DG, and CSG;
Step 2: Make SG according to (1) Fig. 3a;
Step 3: Make DG according to (2) Fig. 3b; and
Step 4: Build the CSG according to (3) and finally, get a completed list array to store this graph Fig. 3c.

3.1.2 Decomposition of the condensed simultaneity graph

The CSG includes the complete information on the dependency and the concurrency of the original ladder diagram. According to this information, the model of the ladder diagram with sequential structure and parallel

Fig. 3 The ladder diagram of the example is represented by condensed simultaneity graph theory. **a** The simultaneity graph. **b** The dependency graph. **c** The condensed simultaneity graph

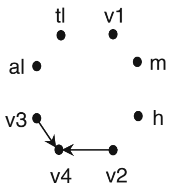
a



v1	→	m	/						
m	→	v1	h	tl	v4	v2	al	v3	/
h	→	m	v4	v2	al	/			
tl	→	m	v4	v2	al	v3	/		
v4	→	m	h	tl	al	/			
v2	→	m	h	tl	/				
al	→	m	h	tl	v4	v3	/		
v3	→	m	tl	al	/				

the simultaneity graph

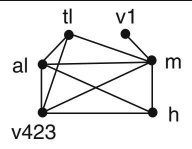
b



v1	→	/	
m	→	/	
h	→	/	
tl	→	/	
v4	→	/	
v2	→	v4	/
al	→	/	
v3	→	v4	/

the dependency graph

c



v1	→	m	/				
m	→	v1	h	tl	v423	al	/
h	→	m	v423	al	/		
tl	→	m	v423	al	/		
al	→	m	h	tl	v423	/	
v423	→	m	h	tl	al	/	

the condensed simultaneity graph

structure can be constructed, which is implemented by decomposition of the CSG.

Once the CSG is obtained, the decomposition of the CSG is an essential step for classifying the components of ladder diagram into sequential structure or parallel structure. The CSG is decomposed into subgraphs via two kinds of decompositions: the connected component decomposition (CCD) and the full connectivity decomposition (FCD).

(1) Connected component decomposition

Any condensed simultaneity graph G can be partitioned into its connected components. Given a graph G=(N, E), where N and E are sets of nodes and edges, the CCD of graph G produces a collection of subgraphs {G1; G2; … Gi…}, such that each subgraph Gi=(Ni, Ei), and no node in Gi is connected to a node in another Gj, where i and j are used to represent the different subgraphs. Performing the CCD on the CSG produces one or more subgraphs to which new sequential structures are assigned. These newly created structures are produced in a sequential order. Since two structures cannot be activated concurrently, they are sequenced according to the order of state variables that appeared in the ladder diagram.

(2) Full connectivity decomposition

A connected graph can be partitioned into a collection of subgraphs via FCD. The FCD of graph G is denoted by FCD (G) and FCD (G) produces a collection of subgraphs {G1; G2; …Gi…}, then every node in Gi is connected to every node in another Gj, where i and j are used to represent the different subgraphs. After accomplishing FCD, the interconnecting edges are removed since all the components in these two subgraphs are connected to each other. Accordingly, components in these two subgraphs can be activated in a parallel way and belong to the parallel structure.

(3) Implementation of CCD and FCD

In order to process the CCD and FCD, data structure is constructed to store the state nodes in the CSG.

```
struct SStateNodeInfo
{
 int                              nID;
 String                           strName;
 int                              nFirstChildID;
 int                              nLeftSiblingID;
 int                              nRightSiblingID;
 int                              nAncestorID;
 bool                             bCollection;
 int                              nLevel;
 String                           strBoolEquation;
 List                             *plistLinkToState;
SStateNodeInfo                    *pNext};
```

Table 1 lists the property specification of this state node structure.

The structure of every state node should be established, and a structure list is constructed to manage all of these structures.

In addition, a default structure is constructed for the original CSG as the head of the structure list:

SStateNodeInfo DefaultStructure {*0*; *NULL*; *0*; *0*; *0*; *0*; *false*; *0*; *NULL*; **plistLinkToFirstState*; **pNext*};

When implementing CCD or FCD, the sequential and concurrent information of every state can be changed by updating the corresponding properties in the structure.

The working flow to decompose the CSG is as follows:

(1) Read the CSG model by adjacent list. Update the property of *listLinkToState* in the structure.
(2) Go through the state nodes, check the property of *listLinkToState*, and select either CCD or FCD is needed.
(3) If CCD is needed, update the structure of this state node with sequential information. Or else, update the structure of this state node with concurrent information.
(4) Repeat Step 2, until the properties of all structures have been updated.

After decomposition, a structure list will be generated with the entire sequential and concurrent information. One structure list for an example of ladder diagram will be shown in the next section.

3.2 Apply CSG theory to optimize the ladder diagram

According to the CSG theory mentioned, it is possible to optimize the organization of the ladder diagram with sequential structure and parallel structure. This section applies the CSG to the ladder diagram shown in Fig. 2 to discuss the optimization. The whole working flow is shown in the Fig. 4.

Step 1: Create the CSG using the definition in the Section 3.1.1 (Fig. 4a).
Step 2: Apply FCD to node "m," the whole graph is divided into two parallel subgraphs as shown in Fig. 4b, {m} and {(v4, v2, v3), vl, h, tl, al}
Step 3: Apply CCD to node "v1." According to the Section 3.1.2 Part (2), the {v1} can be separated from the original graph of {(v4, v2, v3), v1, al, tl, h} as shown in Fig. 4c.

Two subgraphs are divided from the upper level graph. These two subgraphs are sequential. After CCD, they keep the original sequence as executed in the ladder diagram.

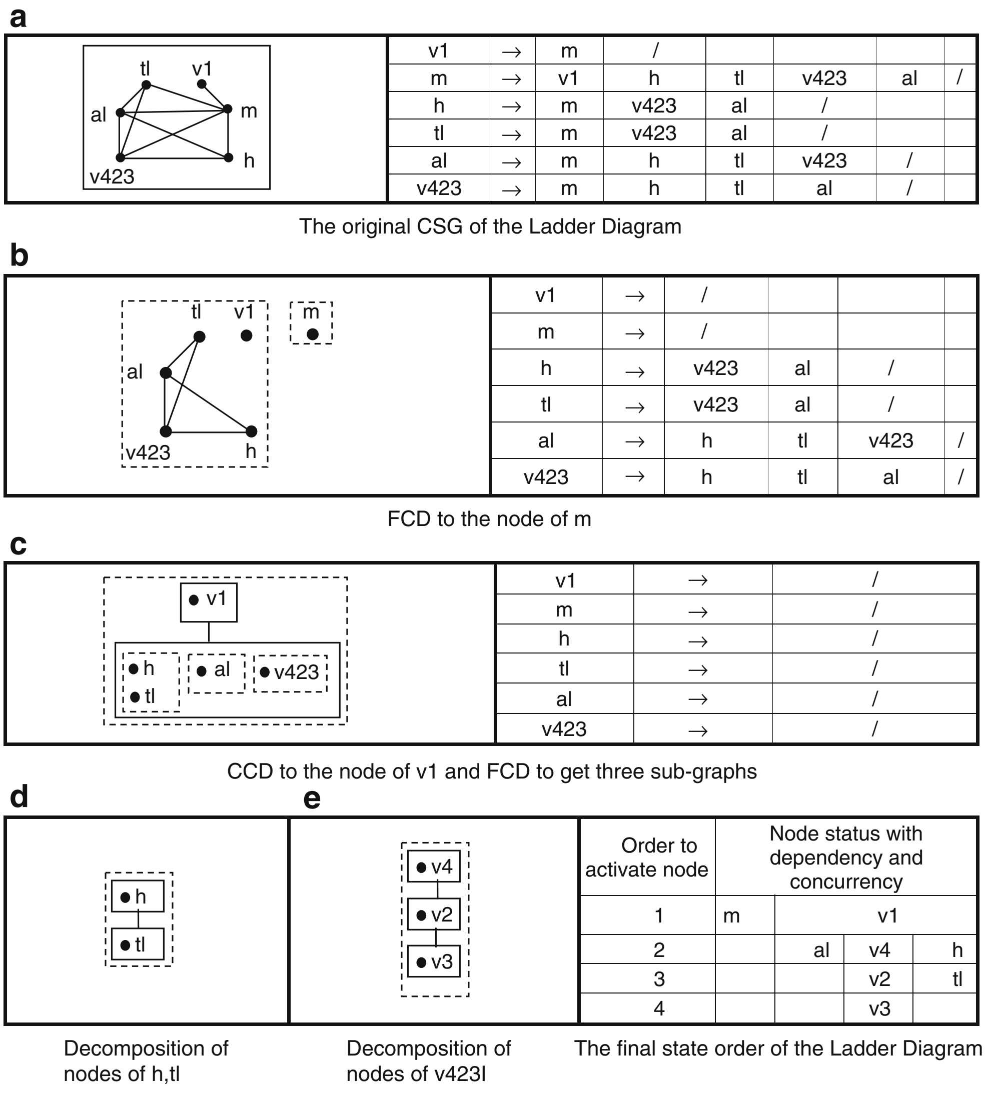
Step 4: Repeat Step 2 and apply FCD to node "al" and "v4, v2, v3." The subgraph {{v4, v2, v3}, al, tl, h} can be partitioned into three subgraphs: {al}, {(v4, v2, v3)}, and {tl, h}. The result of FCD to node "al" and "v4, v2, v3" is shown in Fig. 4c.

These three subgraphs are in parallel level and can be activated at the same time, so they are assigned to three parallel structures.

Step 5: Repeat Step 3 and apply CCD to node "tl" and "h." Subgraph {tl, h} is partitioned into two subgraphs, {tl} and {h}. These two subgraphs are sequential and keep the original sequence as executed in the ladder diagram. The result is shown in Fig. 4d.
Step 6: Since there are no nodes in CSG that can be decomposed, the connected nodes in DG are finally considered. It has been mentioned that DG shows information of rungs depending on the

Table 1 Property specification of state

Property name	Type	Description
nID	integer	Unique identifier assigned to this state node to distinguish it from others
strName	String	Label for this state
nFirstChildID	integer	ID for the first child of this state node
nLeftSiblingID	integer	ID for the left sibling of this state node
nRightSiblingID	integer	ID for the right sibling of this state node
nAncestorID	integer	ID for the parent of this state node
bCollection	bool	If this state node is a condensed node, it's true, else, it's false
nLevel	integer	Considering the parallel structure, the order number to execute this state
strBoolEquation	String	Boolean equation to assign this state variable
*plistLinkToState	list	List of the state connected to this state node in CSG, also the adjacent list of the state node to describe CSG.
*pNext	SStateNodeInfo	Pointer to the next structure


a
tl
v1
al
m
v423
h
The original CSG of the Ladder Diagram
b
tl
v1
m
al
v423
h
FCD to the node of m
c
● v1
● h
● tl
● al
● v423
CCD to the node of v1 and FCD to get three sub-graphs
d
● h
● tl
Decomposition of nodes of h,tl
e
● v4
● v2
● v3
Decomposition of nodes of v423l
The final state order of the Ladder Diagram

v1	→	m	/				
m	→	v1	h	tl	v423	al	/
h	→	m	v423	al	/		
tl	→	m	v423	al	/		
al	→	m	h	tl	v423	/	
v423	→	m	h	tl	al	/	

v1	→	/			
m	→	/			
h	→	v423	al	/	
tl	→	v423	al	/	
al	→	h	tl	v423	/
v423	→	h	tl	al	/

v1	→	/
m	→	/
h	→	/
tl	→	/
al	→	/
v423	→	/

Order to activate node	Node status with dependency and concurrency			
1	m	v1		
2		al	v4	h
3			v2	tl
4			v3	

Fig. 4 **a** The original CSG of the ladder diagram. **b** FCD to the node of m. **c** CCD to the node of v1 and FCD to get three subgraphs. **d** Decomposition of nodes of h, tl. **e** Decomposition of nodes of v423l

outputs of previous rungs during one scan of the ladder diagram. In this step, all of the states that are still not assigned will be lined up in the sequential way. The result is shown in Fig. 4e.

After the last step, a complete model with sequential and parallel structure describing the original ladder diagram has been established. The information on dependency and concurrency is shown in Table 2. This model can perform

Table 2 Information on dependency and concurrency

{{0;	null;	0;	0;	0;	0;	false;	0;	null;	null;	*pNext};
{1;	"m";	0;	0;	2;	0;	false;	1;	"(ls2 # m) & ls1";	null;	*pNext};
{2;	"v1";	3;	1;	0;	0;	false;	1;	"(start # v1) & (! ls2) & (! v3)";	null;	*pNext};
{3;	"al";	0;	0;	4;	2;	false;	2;	"ls2 & as";	null;	*pNext};
{4;	"v4";	6;	3;	5;	2;	false;	2;	"(ls3 # v4) & ls2";	null;	*pNext};
{5;	"h";	7; 4;	0;	2;	false;	2;	"ls2 & (! ts) & (!v3)";	null;	*pNext};	
{6;	"v2";	8;	0;	0;	4;	false;	3;	"ls2 & (! as) & (! v3) & (! v4)";	null;	*pNext};
{7;	"tl";	0;	0;	0;	5;	false;	3;	"ls2 & ts";	null;	*pNext};
{8;	"v3";	0;	0;	0;	6;	false;	4;	"(ls2 & ts & as # v3) & ls1 & (! v4)";	null;	*pNext}};

the same function as the ladder diagram, not only in the sequential way but also in the parallel way. The structure list to represent the state nodes is also shown.

In order to implement FPGA-based PLC with high efficient performance, the model of ladder diagram with the sequential and concurrent information needs to be realized inside FPGA. The VHDL is one of the most popular developing languages for FPGA. So, the next step is to convert this model to VHDL program and implement corresponding sequential and parallel structure in VHDL.

4 Implementation of sequential and parallel operations in FPGA

In Section 3, the original ladder diagram has been analyzed and established a ladder diagram model with sequential and parallel structure. This section explains how to implement this ladder diagram model inside FPGA by converting ladder diagram to VHDL program.

4.1 Implementation of the Conversion from ladder diagram to VHDL program

Usually, while PLC is working, ladder diagram is executed rung by rung from the top to the bottom. However, the FPGA design works in a parallel way once the FPGA device has been configured with the downloaded design. Thus, working principle is very different between the ladder diagram and VHDL program. For some components in ladder diagram, such as normal open and normal close, there are corresponding components in VHDL such as NOT, AND, and OR gates to match each other. For other components such as self-hold and interlock, there is no corresponding component. Therefore, a bridge is needed to eliminate this gap between these two programs. Several researchers used the modeling method like FSM or Petri net as the bridge [5–10, 15]. Actually, a complicated ladder diagram probably consists of thousands of states so that it may easily cause the FSM or Petri net model extreme complex and hard to handle.

In this research, the Boolean equations are chosen as the bridge to connect ladder diagram to VHDL. Boolean algebra is a system of mathematics often used to manipulate logic operation. It's very appropriate to describe logic circuit. As a logic description, Boolean expression almost has the same form as that in VHDL, so it is much easier to convert ladder diagram to Boolean expression than to other models.

Table 3 lists the operator often used in Boolean expression.

To the ladder diagram in Fig. 2, the equivalent Boolean equations are shown in Table 4.

Table 3 The operator used in Boolean expression

!	&	#	$
NOT	AND	OR	XOR

Table 4 Boolean equations for the example in Fig. 2

```
v1 :=(start # v1) & (! ls2) & (! v3)
m :=(ls2 # m) & ls1
h :=ls2 & (! ts) & (!v3)
tl :=ls2 & ts
v4 :=(ls3 # v4) & ls2
v2 :=ls2 & (! as) & (! v3) & (! v4)
al :=ls2 & as
v3 :=(ls2 & ts & as # v3) & ls1 & (! v4)
```

5 Implementation of the sequential operations in VHDL

The sequential operation is critical to assure the correct performance. In the application system of the PLC, some actions must be done after the others, else, it will result serious damage or even the danger to life. This can be controlled over by activating the rungs of the ladder diagram in the correct order. Similarly, because VHDL program works based on process statements, if put, the operation of every dependent rung of the ladder diagram into process statements in VHDL, the sequential operation can be realized by using sensitive signals to sequentially activate different process statements

In order to implement the same sequential operations as the original ladder diagram program, two steps are proposed in this paper: (a) generate sequential sensitive signals, which are used to activate different process states in order and (b) put the operation of every rung into the process statement in VHDL. These two steps can be implemented in VHDL with two types of process statements. The one is the finite state machine process statement. The other is the regular process statement.

In the first step, finite state machine is proposed to generate the sensitive signals. PLC is a discrete event system. Finite state machine is often used to model the discrete event system. So, it is appropriate to control the sequential operation in PLC. In VHDL program, process statements are running in a parallel way. With the sensitive signals generated by finite state machine, process statements can be activated in a sequential way. Finite state machine can be implemented in VHDL as follows (Here, the limit of state status is taken as 8):

```
FSM_process: process(clock,reset)
begin
    if (reset='1') then
        current_process <= "000";
    elsif (clock'event and clock='1') then
        if (current_process ="111")then
            current_process <= "000";
        else
            current_process <= current_process + step_increment;
        end if;
    end if;
end process;
```

This is a process statement for finite state machine. Clock signal and reset signal are the sensitive signals for this process. The signal, current_process, is the sensitive signal which will be used to sequentially activate all of the regular process statements. The variable, step_increment, is used to adjust the order increment of regular process statements to be activated. Usually, it is assigned by “1,” so the regular process statements will be activated one by one. If one of the regular process statements will execute a jump instruction, it will be assigned by a proper value, so the correct regular process statement will be activated in the next clock.

In the second step, the Boolean equations such as those in Table 4 are mapped into regular process statements one by one. Here is an example as follows:

```
Regular_01: process ( current_process )
begin
    if current_ process = "000" then
        V1 <= (start or V2) and ( not V3);
    end if;
end process No_01;

Regular_11: process ( current_process )
begin
    if current_ process = "001" then
        V4 <= (ls3 or V4) and ls2;
    end if;
end process No_11;
```

Where, Regular_00 and Regular_10 mean the label of the regular process statement. One label is corresponding one rung in the ladder diagram program. The signal, current_process, is just the sensitive signal generated by the finite state machine process. In VHDL program, process statements are running in a parallel way. When the sensitive signal is changed, all of the regular process statements will be activated at the same time; but according to the value of the sensitive signal, only the proper regular process statements can be executed at this moment.

With the finite state machine process statement periodically changing the sensitive signal, the sequential cyclic scan in the original ladder diagram can be realized in VHDL program.

5.1 Implement the parallel operations in FPGA

Similarly, the parallel operation is critical to assure high-efficient performance. In the practical application of the control system, some actions can be done at the same time. In the traditional PLC, these actions are executed one by one because of the limit of the microprocessor. However, inside FPGA, they can be implemented by the concurrent sensitive signals. If put, the operations of every independent rung in the ladder diagram program into process statements in VHDL, the parallel operation can be realized by using sensitive signals to activate different process statements at the same time.

The independent rungs are mapped into the regular process statements shown in the following example. Both of these regular process statements will be activated by the sensitive signals, which are generated by the finite state machine process. But it is different with the sequential operations that the regular process statements will be executed at the same time because of the concurrent sensitive signal.

```
Regular_00: process ( current_process )
begin
    if current_ process = "000" then
        m <= (ls2 or m) and ls1;
    end if;
end process No_00;

Regular_01: process ( current_process )
begin
    if current_ process = "000" then
        V1 <= (start or V2) and ( not V3);
    end if;
end process No_01;
```

When the sensitive signal, current_ process, is assigned by “000,” both of these regular process statements are activated and executed at this moment. Accordingly, it means two rungs related to these two process statements in the ladder diagram program are executed at the same time.

6 Case study

To the original ladder diagram program in Fig. 2, there are eight rungs. If this program is executed in sequential way as executed in microprocessor, eight instruction cycles are needed. In FPGA, it only needs four-clock cycles with both sequential operation and parallel operation. The performance becomes more efficient. At the same time, it keeps the same function as the original ladder diagram program.

A converter has been developed to convert the ladder diagram model with information on dependency and concurrency to VHDL program. Once the VHDL program has been generated with the software tools such as the ISE Foundation from Xilinx and the ModelSIM from Mentor Graphics, the converted VHDL program can be debugged and simulated.

In fact, the example of the ladder diagram mentioned in Fig. 2 is the control system for a well-known neutralization system in Fig. 5. The control rules are as follows [14, 15]:

(1) Initially, all the valves are closed, the mixer and heater are off, and the reaction tank is empty. When the start button is pressed, open valve v1 until level sensor ls2 is activated. This fills the tank with the solution to be neutralized.

(2) The following three processes proceed concurrently.

  (a) When the solution level rises above level sensor ls2, start mixer m. When the level drops below ls1, stop the mixer.

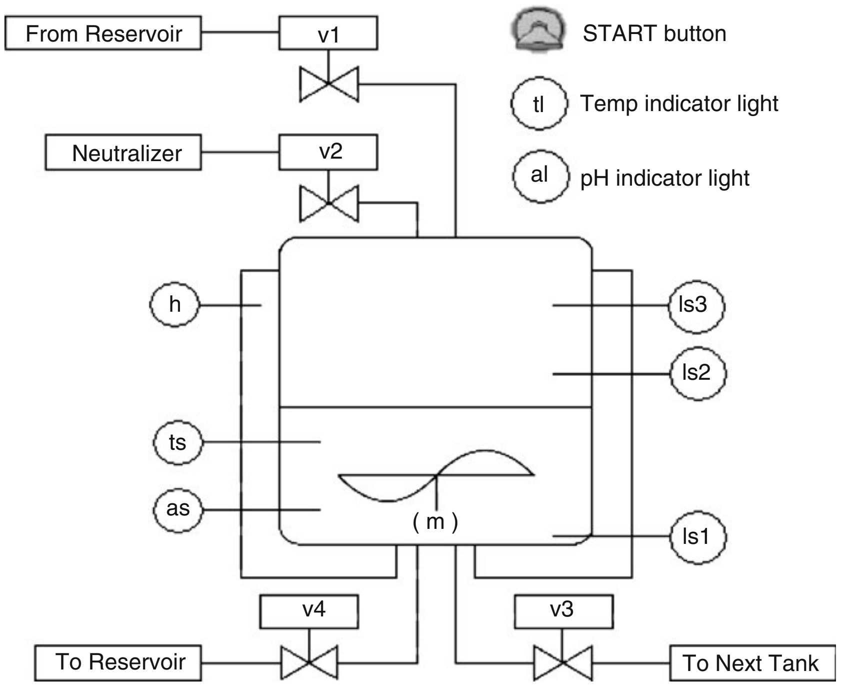


Fig. 5 Chemical vat neutralization system

(b) Whenever the temperature of the solution is below a preset point, energize heater h.

(c) Whenever the pH of the solution is unbalanced, open valve v2 to add the neutralizer.

(3) If the tank becomes full, as indicated by the activation of level sensor ls3, close v2 to stop the in flow of the neutralizer. Next, open valve v4 to reduce the level of the solution to the point indicated by ls2. Then close v4 and proceed with step (c).

(4) When both the temperature and pH of the solution are correct, de-energize the heater and close v2. Then open valve v3 to drain the tank. When the tank is empty, as indicated by the deactivation of ls1, close v3 and proceed with step (1).

(5) Two indicator lights, tl and a1, are regulated. Light tl should turn on whenever the solution level is above ls2 and the temperature has reached the preset value. The same rule is used to light al and pH.

With the ModelSIM, the VHDL program can be simulated and the result is shown in Fig. 6.

From the Fig. 6, the following results can be verified:

In the first four-clock cycle, when the start button is pressed, if both ls1 and ls2 are on, the mixer and the heater will turn on, v2 will be opened to add the neutralizer.

In the second four-clock cycle, if ls3 is on, the solution reaches level 3. v4 will be opened and v2 will be closed. It means open the valve 4 for reducing the level of the solution and at the same time close the valve to stop adding the neutralizer.

In the third four-clock cycle, if the level of the solution goes down under level2, ls2 and ls3 are off, valve v1 will be reopened to add solution. Before the level of solution reaches level 2, the heater turns off. And the valve 4 should be closed.

In the fourth four-clock cycle, if the level of the solution return back to level 2, close the valve 1, turn on the heater, open the valve 2 for neutralizing.

In the fifth four-clock cycle, once the temperature and pH are satisfied (ts and as turn to high), tl and al turn on. The valve 2 is closed and the heater is off. Open the valve 3 to drain the tank.

The results of simulation verify that the converted VHDL program can perform the same function as the ladder diagram does. Synthesize this VHDL program and download it to the FPGA, the control system will be implemented inside FPGA.

7 Conclusions

In order to overcome the performance limitation of the traditional microprocessor-based PLC and dramatically improve the PLC performance, in this paper, FPGA-based PLC is proposed. With FPGA device, it is possible to implement the ladder diagram in programmable hardware solution. In order to efficiently implement the ladder diagram inside FPGA, CSG optimization method is introduced to reorganize ladder diagram with sequential and parallel structure. Two important components in VHDL design, finite state machine scheme and sensitive signal, make sure to implement the original ladder diagram in VHDL design not only in a sequential way but also in a parallel way.

The authors developed the software to convert the ladder diagram to VHDL design. Using this converter, a case

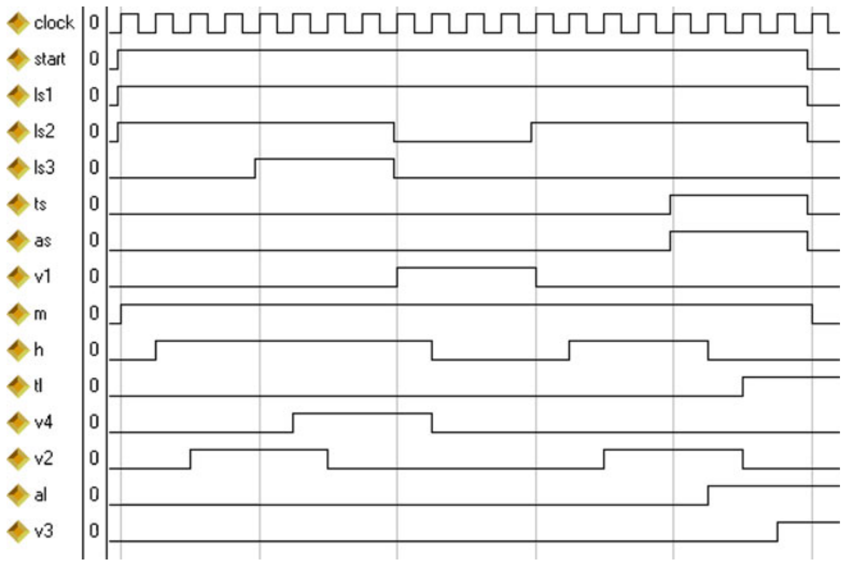


Fig. 6 The logic sequence of the example implemented inside FPGA

study was made to verify that the proposed approach to implement FPGA-based PLC is feasible, and the PLC performance is improved dramatically. Furthermore, as a programmable hardware solution, FPGA device is reconfigurable, which make it easier for the application system to be modified and maintained. The design method is more flexible.

Acknowledgment The authors wish to express their sincere appreciation for the generous support from Mori Seiki Co. Ltd., which made this work possible.

References

1. Auslander DM, Pawlowski C, Ridgely J (1998) Reconciling Programmable Logic Controllers with Mechatronics Control Software, Proceedings of the 1998 IEEE international Conference on Control Applications, pp 415–420
2. Ikeshita M, Takeda Y, Murakoshi H, Funakubo N, Miyazawa I (1999) Application of FPGA to high-speed programmable controller - development of the conversion program from SFC to Verilog IEEE Symposium on Emerging Technologies and Factory Automation, IEEE, Piscataway, NJ, USA, ETFA.v2:1386–1390
3. Miyazawa I, Nagao T, Fukagawa M, Itoh Y, Mizuya T, Sekiguchi T (1999) Implementation of ladder diagram for programmable controller using FPGA. IEEE Symposium on Emerging Technologies and Factory Automation, IEEE, Piscataway, NJ, USA, ETFA. v2:1381–1385
4. Chen J, Patyra MJ (1994) VHDL modeling of a multivariable fuzzy logic controller hardware system IEEE International Conference on Fuzzy Systems. IEEE, Piscataway, NJ, USA, v1:129–132
5. Abdel-Hamid AT, Zaki M, Tahar S (2004) A tool converting finite state machine to VHDL, Canadian Conference on Electrical and Computer Engineering Canadian Conference on Electrical and Computer Engineering; Technology Driving Innovation, 4:1907–1910
6. Kuusilinna K, Lahtinen V, Hamalainen T, Saarinen J (2001) Finite state machine encoding for VHDL synthesis. Comput Digi Tech 148:23–30
7. Adamski M, Monteiro JL (2000) From interpreted Petri net specification to reprogrammable logic controller design. IEEE International Symposium on Industrial Electronics 1:13–19
8. Adamski M (1998) SFC, Petri nets and application specific logic controllers, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. IEEE, Piscataway, NJ, USA, 1:728–733
9. Uzam M, Jones AH (1998) Discrete event control system design using automation Petri nets and their ladder diagram implementation. Int J Adv Manuf Technol 14(n10):716–728
10. Lee JS, Hsu PL (2005) An improved evaluation of ladder logic diagrams and Petri nets for the sequence controller design in manufacturing systems. Int J Adv Manuf Technol 24(n3–4):279–287
11. Welch JT (1992) Translating unrestricted relay ladder logic into Boolean form. Computers in Industry 20:45–61
12. Shanta S, Dipali S (2005) A new generation of PLC—an FPGA based PLC. Proceedings of the SICE Annual Conference, SICE 2005 Annual Conference in Okayama—Proceedings. pp 2367–2370
13. Yadong L, Kazuo Y, Makoto F, Masahiko M (2005) Model-driven programmable logic controller design and FPGA-based hardware implementation. Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference—DETC2005, 4:81–88
14. Falcione A, Krogh BH (1992) Design recovery for relay ladder logic, the first IEEE conference on control applications. Dayton, OH, pp 90–98
15. Lee JI, Chun SW, Kang J (2002) Virtual prototyping of PLC-based embedded system using object model of target and behavior model by converting RLL-to-state chart directly. J Systems Archit 48:17–35