Paper

An FPGA implementation of hard-wired sequence control system based on PLC software

Shuichi Ichikawa* Non-member Masanori Akinaka* Non-member
Hisashi Hata* Non-member Ryo Ikeda* Non-member
Hiroshi Yamamoto* Non-member

Although a Programmable Logic Controller (PLC) has been widely adopted for the sequence control of industrial machinery, its performance does not always satisfy the recent requirements in large and highly responsive systems. With the state-of-the-art FPGA technology, it is possible to implement a control program with hard-wired logic for higher response and reduced implementation cost/space. This approach is also worthwhile for transmigration of legacy PLC software into forthcoming FPGA-based control hardware.

This study presents a systematic method to implement a hard-wired sequence control from PLC software. PLC instructions are converted into VHDL codes, and then implemented as logic circuit with various peripheral functions. Productive PLC programs were examined with Mitsubishi Electric FX2N PLC and Altera Stratix II FPGA, and were shown to fit into a common FPGA chip. A straightforward Sequential design was estimated to be 184 times faster than PLC, while a performance-oriented Flat design was estimated to be 44 times faster than Sequential design (i.e., 8050 times faster than PLC). A practical perfect layer winder system was actually built and successfully operated with our FPGA control board, whose logic design was implemented with our tools.

Keywords: Programmable Logic Controller (PLC), Field Programmable Gate Array (FPGA), sequence control, ladder diagram, embedded systems

1. Introduction

A Programmable Logic Controller (PLC) has been widely adopted for the sequence control of industrial machinery. Since industrial machinery is customized for each customer and operates for a long period, there exists enormous number of legacy software for PLC systems. It is practically very important to make full use of these legacies for the current and future systems.

Although the importance of PLC is definite and undoubted, some problems have arisen recently. The first problem is that the performance of PLC does not always satisfy the recent requirements in large and highly responsive systems. Another problem of PLC is that a PLC program is easy to duplicate and to analyze. This often results in the leakage of valuable trade secrets and the rise of clone products. As a solution to these problems, a hard-wired implementation of PLC program is proposed in this study.

Field Programmable Gate Array (FPGA) is a kind of reconfigurable LSI, which can be programmed at all times by downloading its configuration data. By implementing a control program with hard-wired logic using an FPGA device, a flexible and highly responsive system could be realized. Maximally ten million logic gates are available in a single FPGA chip, which is enough to implement a very large control system. The FPGA implementation might lead to downsizing and reduction of system components.

* Department of Knowledge-based Information Engineering, Toyohashi University of Technology. 1-1 Hibarigaoka, Tempaku, Toyohashi 441-8580 Japan. (Email: ichikawa@ieee.org)

It should be also noted that FPGA is more secure than PLC in protecting intellectual properties, because it is more difficult to analyze an FPGA design than to analyze a PLC program. Some recent FPGA devices provide design security features (e.g., Altera Stratix-II [1]), which protect the design by cryptographic encryption algorithm.

For all these merits, there are some drawbacks in an FPGA-based control system. It is necessary to re-generate the circuit from the PLC software, whenever the control program is updated; it is thus not suited for the application where the program is frequently changed. Reliability and noise immunity issues are also practical concerns in actual control systems, since recent FPGA devices are driven at a low voltage (e.g., 1.8 V). The authors *never* insist on replacing all conventional PLCs with FPGAs. Rather, we suggest that FPGA technology might offer a promising alternative solution for some applications, particularly for highly responsive systems.

This study presents a systematic method to implement a hard-wired sequence control based on PLC software. First, a PLC program (instruction sequence) is translated into logic description in VHDL [2] with our converter. To support various PLC instructions and peripheral devices, a control logic library was compiled from the corresponding VHDL routines. All these VHDL source codes are then synthesized, placed, and routed for a target FPGA device by CAD software. The authors developed a new FPGA control board with various I/O devices, and a perfect layer wider system was actually implemented with this FPGA board.

The rest of this paper is organized as follows. Section 2 outlines the background and related studies of this study.

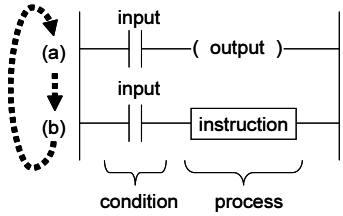


Fig. 1. Overview of ladder diagram. [3] (©2006 IEEE)

Section 3 describes the method to translate PLC programs to hardware descriptions. The evaluation results of two sample PLC programs are detailed in Section 4. Section 5 summarizes the overall design framework, and Section 6 introduces experimental industrial machinery that was implemented with an FPGA control board. Two supplementary topics are then examined in Section 7. Section 8 concludes the paper, showing the list of items left for future studies.

2. Background and Related Studies

There have been some studies on the implementation of a control program in FPGA. For example, Adamski and Monteiro [4] [5] presented a design methodology that translates "interpreted Petri net specification" into hardware description languages. Wegrzyn et al. [6] [7] presented a framework that transforms rule-based descriptions (e.g., interpreted Petri net) into logic descriptions (e.g., VHDL). Ikeshita et al. [8] presented a conversion program that translates SFC (Sequential Function Chart) description into Verilog-HDL for logic synthesis. Silva et al. [9] [10] presented a hardware-software platform to implement logic controllers with both PLC and FPGA, where Petri net descriptions are translated into VHDL codes.

All these studies concentrated on techniques to convert functional-level control programs into logic circuits. In contrast, this study deals with control programs at the lowest level: PLC instruction sequence. Although it is more difficult to analyze, our scheme would be applicable to a wider area of control programs. Moreover, our technique might be expendable to the instruction sequence of various embedded processors.

Miyazawa et al. [11] proposed a method to translate PLC programs of ladder diagram (LD) into VHDL programs. Welch and Carletta [12] proposed an FPGA architecture, which implements relay ladder logic directly. Du et al. [13] presented an optimization technique in LD-VHDL conversion.

Though the ladder diagram is almost equivalent to a PLC instruction sequence, the above-mentioned studies only examined very fundamental logic functions such as AND, OR, NOT, and flipflop, while providing no detailed discussion about actual PLC applications. To the contrary, the present study deals with advanced features of PLC that are required in real-world applications, and presents quantitative evaluation results.

A preliminary version of this study was presented in IEEE ISIE 2006 [3]. Major differences from the preliminary work are summarized below.

- The converter was improved to generate better logic designs, which require fewer cycles for each scan by reducing unnecessary states. The effects of this improvement are perceived by comparing Tables 2 and 3 to the Tables II and III of the previous study [3].
- Evaluation platform was updated from an old APEX20KE device [14] to a new Stratix II device [1].
- Evaluation results were thoroughly updated with the new converter and new FPGA platform.
- Productive industrial machinery was actually built with an FPGA controller, whose logic design was generated by our converter from PLC program.

The advantages of FPGA implementation of PLC program were also presented by C. Economakos and G. Economakos in the recent publications [15] [16]. More comments on their works are found in Section 7.1.

3. Translation of PLC Program to Hardware Description

3.1 Ladder Diagram The ladder diagram has been widely accepted to describe PLC programs. A ladder diagram consists of one or more *rungs*, each of which consists of a condition part and a process part (Fig. 1). Either the condition part or the process part can be an input/output (a) or an instruction (b). The output of a rung is activated if the corresponding input condition is satisfied; otherwise, the output is deactivated. The instruction of a rung is executed if its input condition is satisfied. Rungs are ordered, and interpreted in due order.

A ladder diagram is executed in the following manner:

(1) At the beginning of a ladder, all inputs are collected and stored into the corresponding internal memory elements, which are read and modified by rungs (**Input phase**).

(2) Rungs are interpreted in due order (**Execution phase**).

(3) When the bottom of a ladder is reached, all output ports are updated by the corresponding internal memory values (**Output phase**).

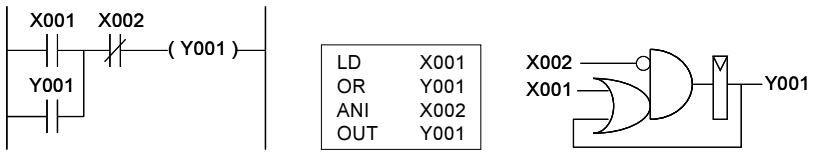
(4) The ladder is then executed all over again from the input phase.

Repeated execution of the above-mentioned cycles is called *cyclic scan*, and the period of cyclic scan is called *scan time*. By making the scan time shorter, the system becomes more responsive.

3.2 Translation of a Rung Figure 2 (a) illustrates a rung of a ladder diagram for Mitsubishi Electric FX2N PLC [17], which is adopted as an evaluation platform in the following discussion. A distinct advantage of FX2N PLC is that its instruction set specifications are open to the public [18].

FX2N instruction set includes 160 instructions with various types of operands: e.g., switch X, coil Y, internal relay M, data register D, constant K, and timer T. In Fig. 2, the switches X001 and X002 correspond to start switch and stop switch, respectively. The slash on X002 denotes negative logic. If X001 is on and X002 is off, the output coil Y001 is turned on; this results in X001 bypassed, and thus Y001 holds while X002 is off. If X002 is turned on, Y001 is inactivated. This logic is called *self-holding logic*. The rung (a) is translated into the instruction sequence shown in Fig. 2 (b), while this control logic can be translated into the corresponding logic circuit (Fig. 2 (c)).

Another example is shown in Fig. 3, where the process part



(a) Ladder diagram (b) Instruction sequence (c) Logic circuit

Fig. 2. An example of FX2N PLC program: self-holding logic. [3] (©2006 IEEE)

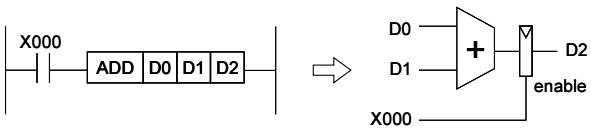


Fig. 3. Another example: arithmetic instruction. [3] (©2006 IEEE)

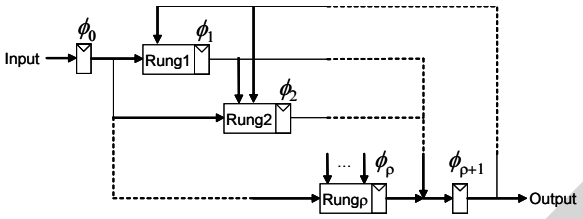
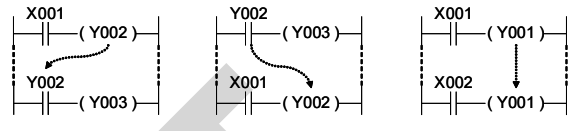


Fig. 4. Sequential Design.



(a) Data dependence (b) Anti-dependence (c) Output dependence (double coil)

Fig. 5. Three dependencies to be considered. [3] (©2006 IEEE)

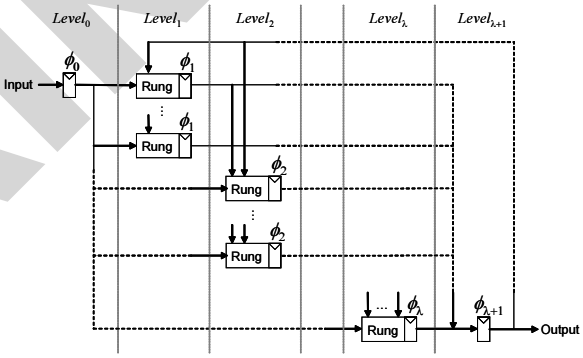


Fig. 6. Levelized Design.

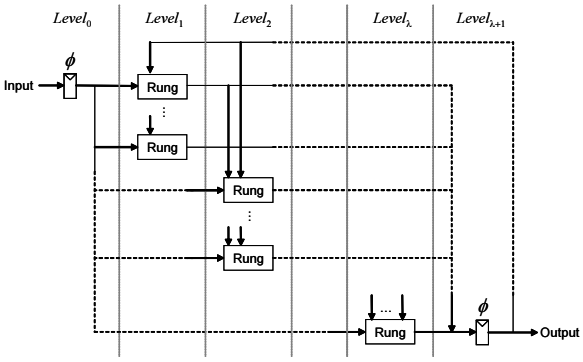


Fig. 7. Flat Design.

of a rung is an arithmetic instruction. In this case, an ADD instruction is translated into an adder, whose output is captured by a register if the corresponding condition is satisfied.

Our experimental converter generates a VHDL source code from an instruction sequence of FX2N. The supported instructions include 23 basic programming instructions (out of 27) and 13 applied instructions (out of 133), which are summarized in Table 1. This list is not long, but includes enough instructions for the following evaluations. The authors are still extending the list of supported instructions, basically on a demand-driven basis to support real-world control programs.

3.3 Sequential Design To generate a logic circuit that literally simulates a whole ladder program, it is straightforward to design a sequential circuit, which activates one rung for each cycle in due order. This design is illustrated in Fig. 4, and is designated by *Sequential design* in the following discussion.

Although Sequential design reproduces the exact behavior of a ladder program, it requires ρ+2 cycles for each scan. Here, ρ is the the number of rungs of a ladder, and two additional cycles are required for input phase and output phase described in Section 2. The circuit is driven by (ρ+2)-phase non-overlapping clocks ($\phi_0, ..., \phi_{\rho+1}$).

For further reduction of scan time, it is essential to utilize parallelism in the control program.

3.4 Levelized Design It is possible to execute two or more rungs in parallel, as a superscalar microprocessor does, if dependencies among rungs are properly maintained. Figure 5 (a) shows an example of data dependence, where the output of the upper rung is referred by the lower rung. In Fig. 5 (b), the input of the upper rung is overwritten by the lower rung (anti-dependence). Figure 5 (c) shows an example of output dependence, or double coiling [17], where the output of the upper rung is overwritten by the lower rung. In any of these cases, it is essential to execute the upper rung before the lower rung to derive the same result as in the original ladder diagram.

Dispatching each rung to the earliest cycle possible (while keeping all dependencies), we can reduce the clock cycles required for each scan. In this study, we simply levelize rungs according to their dependencies, as in levelized compiled code simulation of a logic circuit [19]. The rungs that have no preceding rungs are labeled by $Level_1$, and the rungs that are dependent on $Level_i, Level_j, ...$ are labeled

Table 1. Supported instructions.

Category	Mnemonic
Basic instructions	LD (LD=), LDI, LDP, LDF, AND (AND=), ANI, OR, ORI, ANP, ANF, ORP, ORF, ANB, ORB, MC, MCR, OUT, SET, RST, PLS, PLF, NOP, END
Applied instructions	MOV (DMOV), ADD (DADD), SUB (DSUB), MUL (DMUL), DIV (DDIV), TO (DTO), FROM (DFROM), BMOV, WAND, ROL, ZRST, HEX, INC

by $Level_{\max(i,j,\ldots)+1}$. We can emulate a ladder by a sequential circuit, which activates the input phase at $Cycle_0$, the rungs of $Level_i$ at $Cycle_i$, and the output phase at $Cycle_{\lambda+1}$, where λ is the maximal level of rungs. Thus, the number of cycles for each scan would be $\lambda + 2$ in this circuit. This design is illustrated in Fig. 6, which is designated by *Levelized design* in the following discussion.

3.5 Flat Design Sequential design activates the circuit of each rung, one for each cycle, in due order. Levelized design activates the circuit block of each level, one for each cycle, from upstream to downstream. This brings up the question of why it is implemented by a sequential logic circuit at all. It is not necessary to split the execution phase into cycles, because the inputs and outputs are updated only at the end of each scan.

In fact, it is possible to implement the execution phase by a combinatorial logic circuit. Let us examine Levelized design as an example. The input data of $Level_j$ circuit are fed from $Level_i$ $(0 \leq i < j)$ in Levelized design. When the process part of a rung at $Level_j$ is an output, we can simply remove the internal memory element of this output, and feed data downstream by wire. When the process part is an arithmetic instruction, we have to replace the memory element by a multiplexer, which feeds data downstream. Since the instruction might or might not take place depending on the value of its condition part, a multiplexer is required to select either the new value (generated at $Level_j$) or the original value (fed from $Level_i$). The execution phase could thus be converted into a combinatorial logic circuit.

The input and output phases are also redundant. In Sequential and Levelized designs, two cycles are consumed for input and output phases. Since the inputs are always updated just after the outputs, the input phase and output phase can be unified to one cycle.†

Taking these two ideas together, each scan could be performed in one cycle. The derived design is illustrated in Fig. 7, which is designated by *Flat design* in the following discussion. Flat design is expected to be faster than Levelized design for the following reason. The scan time of Levelized design t_l is given by $t_l = (\lambda + 2) \max_i \delta_i$, where δ_i is the maximal delay of $Stage_i$ (the circuit of $Level_i$). Assuming that the difference in logic is negligible, the scan time of Flat design t_f is expected to be shorter than t_l, because $t_f \leq \Sigma_i \delta_i \leq \lambda \max_i \delta_i < t_l$ holds. In many cases, Flat design would be much faster than Levelized design, because $t_f \ll \Sigma_i \delta_i$ usually holds. Another advantage of Flat design is that CAD software is generally good at optimizing combinatorial logic circuits, compared to sequential circuits.

3.6 Resource Restriction Though scan time is very important, the logic scale of control circuit is equally important for practical applications. Particularly in translating a large control program, it is essential to restrict resource usage. In this section, we discuss the circuit generation with resource restriction.

In the above-mentioned designs, each instruction is translated into its hardware counterpart, and the consequent circuit contains as many components as instructions. A logic operation does not cost much, because it is implemented by a bitwise circuit. Meanwhile, an arithmetic instruction matters very much, because it requires a 16-bit or 32-bit wide arithmetic unit. Thus, it is very important to restrict the number of arithmetic units by sharing them among instructions.

In Sequential design, it is very easy to share arithmetic units, because only one rung is activated in each cycle. It is enough to generate one arithmetic unit for each kind of arithmetic operation, if its input is multiplexed and its output is redirected properly. In the following discussion, this design is designated by *shared* arithmetic units. It should be noted that a shared design might incur a significant amount of hardware for the input multiplexers and output interconnects in exchange for reduction of arithmetic units. The original design, which generates as many arithmetic units as arithmetic instructions, is designated by *dedicated* in the following discussion.

In Levelized design, resource limitations of an arithmetic unit may affect the scheduling of rungs, which can result in the increase of clock cycles. It is thus necessary to choose a good resource scheduling algorithm to minimize the scan time. This problem is a type of multiprocessor scheduling problem [20] [21] which is generally difficult to solve. Though a simple list scheduling was implemented in this study, other algorithms should be investigated in future studies.

3.7 Optimization Issues Though there are many possible optimizations, we left most of them for future studies, and concentrated on evaluating the fundamental aspects of this method. The following are some items for future attention.

The first issue is the optimization of instruction sequence. In this study, our converter literally translates an instruction sequence into the corresponding logic description. However, it is possible to generate a better logic circuit by analyzing and rewriting the instruction sequence. For example, the sum of four values a, b, c, d can be calculated by either $((a+b)+c)+d$ or $(a+b)+(c+d)$. Literally converted, the former would result in a cascade of three adders, while the latter would be a balanced tree of three adders. Generating a balanced tree of adders from the former instruction sequence is left for future studies.

Area-Time trade-off is another important issue. Although some results are shown in the following evaluation, automatic exploration of the best trade-off is beyond the scope of this work.

† Strictly speaking, the unification of input phase and output phase may cause trouble, wherever there exists an external loopback from output to input. Such design is rather exceptional, and should be avoided to assure high performance.

Table 2. Evaluation results of a PID controller program.

Device	Design	Arithmetic unit	Num. of states	Max. Freq. [MHz]	Logic Scale [ALUT]	Memory [bit]	DSP elements	Scan time [s]
PLC	–	–	–	–	–	–	–	7.98×10^{-4}
FPGA	Sequential	dedicated	12	108.57	486	0	24	1.11×10^{-7}
		shared $\times 1$	12	87.63	479	0	8	1.37×10^{-7}
	Levelized	dedicated	9	113.51	480	0	24	7.93×10^{-8}
		shared $\times 1$	9	88.32	475	0	8	1.02×10^{-7}
	Flat	dedicated	1	61.90	448	0	24	1.62×10^{-8}

Table 3. Evaluation results of a sample ladder program.

Device	Design	Arithmetic unit	Num. of states	Max. Freq. [MHz]	Logic Scale [ALUT]	Memory [bit]	DSP elements	Scan time [s]
PLC	–	–	–	–	–	–	–	1.61×10^{-3}
FPGA	Sequential	dedicated	74	8.47	5850	1280	56	8.74×10^{-6}
		shared $\times 1$	74	6.50	2859	1280	8	1.14×10^{-5}
	Levelized	dedicated	12	8.00	5681	0	56	1.50×10^{-6}
		shared $\times 1$	17	6.81	2704	0	8	2.50×10^{-6}
		shared $\times 2$	14	6.48	3961	0	16	2.16×10^{-6}
		shared $\times 3$	13	6.35	5010	0	24	2.05×10^{-6}
		shared $\times 4$	12	6.57	6309	0	32	1.83×10^{-6}
	Flat	dedicated	1	5.00	4624	0	56	2.00×10^{-7}

4. Evaluation Results

This section presents some evaluation results of two sample PLC programs. The evaluation flow is shown by broken lines in Figure 8. First, the scan time of PLC (H) is estimated from its instruction sequence (B) according to the execution time for each instruction [17]. Since the execution time of each instruction is dependent on the value of the corresponding condition part, the worst case scan time is estimated in this evaluation. The PLC instruction sequence is then translated into the hardware description in VHDL by our translator (C). This VHDL description is processed by Altera Quartus II 6.0 SP1 software to generate an FPGA design for Altera Stratix II FPGA [1]. In this study, the target device was set to EP2S60F672C5ES with 48352 ALUTs (adaptive look-up tables), 2.4 Mbit RAM, and 288 DSP elements.†† The optimization options of Quartus II were set to default. The scan time of FPGA (I) is calculated by dividing the number of states by the estimated maximum operational frequency of the circuit.

4.1 PID controller Table 2 summarizes the evaluation results of a PLC program which implements a simple PID controller with 32-bit fixed point arithmetic. The PID controller is classical, but has been frequently used in many control applications to date. This PLC program includes 23 instructions, which include 4 add/subtract instructions and 3 multiply instructions.

FX2N PLC takes 798 μsec for each scan of PID code, in which 508 μsec is consumed by END instruction. Since the END instruction is placed to finish the current scan and to carry out the process of updating outputs and inputs, 64% of PLC scan time is consumed to update inputs and outputs. In contrast, FPGA implementations can perform input/output phases in one or two cycles, which results in very high responsiveness.

Sequential (dedicated) design is approximately 7200 times faster than FX2N PLC, yet requires only 486 ALUTs and 24 DSP elements. DSP elements are used here as the building blocks of multipliers, which support multiply instructions. Levelized (dedicated) design is 1.4 times faster than Sequential (dedicated) design, although the logic scale is almost the same. Flat design is 4.9 and 6.9 times faster than the Levelized (dedicated) and Sequential (dedicated) design, respectively.

In Table 2, "shared $\times n$" designates a design that includes maximally n arithmetic units for each kind of arithmetic operation. Though it is possible to set a different limitation for each operation (e.g., 2 for adder and 1 for multiplier), we applied the same limit to all kinds of arithmetic operations in this experiment.

In both Sequential (shared $\times 1$) and Levelized (shared $\times 1$) designs, the usages of DSP elements were reduced from 24 to 8. This is rational, since the dedicated design includes three multipliers that correspond to three multiply instructions in the original PLC program, while the shared $\times 1$ design implements a single multiplier. Although four adders are also reduced to one, this reduction is canceled by the increase of input multiplexers. Thus, the usage of ALUTs is almost equal for the dedicated and shared $\times 1$ designs.

It should be noted that the operational frequency may be decreased by sharing resources, since additional components are inserted in the data-path. In case of PID controller, the respective overhead for resource sharing is 25% (Sequential) and 29% (Levelized) in scan time.

The designs with more resources (shared $\times 2$ or more) were not examined for PID controller, since there is no resource competition with a single arithmetic unit for each arithmetic operation.

4.2 Practical Control Program Table 3 lists the evaluation results of a sample PLC program, which was derived from an actual product. This PLC program includes 165 instructions, which include 6 add/subtract instructions, 12 multiply instructions, and 9 divide instructions.

Sequential (dedicated) design is approximately 184 times faster than FX2N PLC. Although Sequential (dedicated) and Levelized (dedicated) designs are almost the same in maxi-

†† One DSP element correspond to a 9x9 multiplier, and eight DSP elements correspond to a 36x36 multiplier. [1]

mum operational frequency and logic scale, Levelized (dedicated) design is 5.8 times faster than Sequential (dedicated) design. Flat design is even 7.5 times faster than Levelized (dedicated) design, yet its ALUT usage is 19% smaller. Compared to PLC, Flat design achieves 8050 times higher performance, using only 4624 ALUTs and 56 DSP elements.

As stated in Section 3.6, shared designs require more cycles than the corresponding dedicated design for each scan. The operational frequencies are also smaller than that of the dedicated design. All these result in larger scan times of shared designs.

However, the reduction of hardware resources is larger in this program than in the PID controller. Sequential (shared $\times 1$) achieved 51% reduction of ALUTs, while reducing 86% of DSP elements. Levelized (shared $\times 1$) design also achieved 52% reduction of ALUTs, while reducing 86% of DSP elements. Levelized (shared $\times 2$ or more) were faster than Levelized (shared $\times 1$) accordingly, in exchange for increased logic resources.

Sequential designs are distinct from other designs in memory usage, since the embedded RAM modules are utilized only in Sequential designs. These RAMs are not explicitly used in any designs; they are adopted automatically by the logic synthesizer.

In logic synthesis, a number of registers might be aggregated into a single memory module, if these registers are not accessed simultaneously. Since the embedded RAMs are highly integrated, the implementation cost might be reduced by aggregating many registers into RAMs. The logic synthesizer automatically chooses registers or RAMs for implementation as the case may be.

In Sequential designs, only one output register is written in each cycle, while most registers are only read a few times in each scan. As a result, the logic synthesizer adopted embedded RAM for implementation. In Levelized designs, more rungs are processed in parallel, and thus more registers are read and written in parallel. Consequently, registers were chosen for implementation by the logic synthesizer. Such choice is dependent on each individual case; in fact, no RAMs were adopted in the PID controller implementations (cf. Table 2).

5. Design Framework

Figure 8 illustrates the framework of tools available to translate, integrate, and implement the logic circuit of a control system onto an FPGA. In Fig. 8, double rectangles designate three tools implemented by the authors. The solid-line arrows designate the path to generate logic circuit from control logic, while the dotted-line arrows designate the path for performance evaluation.

A ladder program is designed with Mitsubishi GX Works software (step A in Fig. 8), and then translated into an instruction sequence with GX Converter (B). As described in Sect. 3, our translation tool translates the instruction sequence into the hardware design described in VHDL (C). Users may write the top-level design (F) by themselves, or may prepare the interface description file (E) instead. The top-level design can be generated from the interface description by our interface logic generation tool. In this example, the top-level design includes a library component STPG,

Fig. 9. SPM05-02: a perfect layer winder system with an FPGA control board.

which is a peripheral device of FX2N PLC.

Even if a PLC program could be translated into hardware, it does not work without peripheral devices. To achieve a higher level of integration, it is essential to integrate peripheral devices on an FPGA together with the control logic circuit. Thus, the authors prepared a control logic library, which includes (1) various components to replace peripheral devices of PLC, (2) template circuits that correspond to PLC instructions, and (3) some support functions. In this example, a programmable pulse generator STPG (D) is extracted from the peripheral library. Finally, Altera Quartus II software processes a top-level design (F), a control logic (C), and a peripheral component (D) to generate a bitstream, which is downloaded onto a target FPGA (G).

It is worth integrating a microprocessor core into the FPGA, because actual control systems include many parts that are not necessarily implemented in a logic circuit, e.g., user interface, network communication, etc. In such a case, most soft real-time tasks would be handled by a microprocessor, while hard real-time tasks are translated into a custom circuit, which might be attached as a peripheral device of a microprocessor.

Another prospective option is the use of C-based hardware design tools, which provides a higher-level of abstraction. Some of the topics discussed in this paper, e.g., dependency analysis and resource scheduling, might be handled by those higher-level tools.

Meanwhile, in this study, conventional VHDL tools were adopted by the following reasons.

- When we began this project, C-based tools were still immature and did not fulfill our requirements.
- We had to control the designs directly to evaluate the various design options quantitatively, instead of leaving them to design tools.

C-based design tools will be a practical choice hereafter, since they are rapidly improving in quality and reliability.

6. Experimental System

The authors developed a "perfect layer winder" (Fig. 9) in conjunction with Yashima Netsugaku Co., Ltd. [22], which is a specialty manufacturer of various fiber winders. This winder has practical specifications for a fully automated operation;

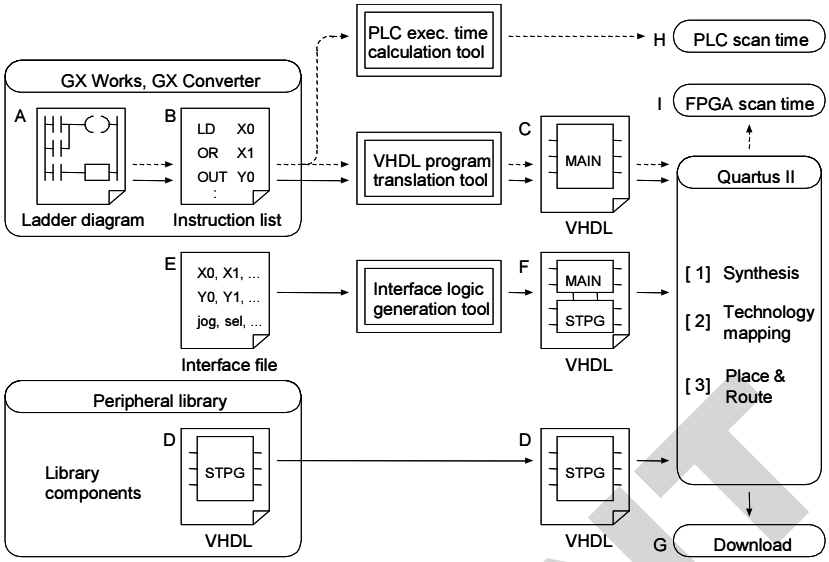


Fig. 8. Design framework. [3] (©2006 IEEE)

Table 4. The specifications of TUTFA-CYCII-FI.

FPGA device	Altera Cyclone II EP2C8F256 (EP2C20F256 is also available)
Inputs	25 ch. (photo-isolated), 4 ch. (high-speed, photo-isolated), 4 ch. (differential, SN75175)
Outputs	40 ch. (photo-isolated, open-collector), 4 ch. (high-speed, photo-isolated, open-collector), 4 ch. (differential, SN75174), 2 ch. (analog, DC±10V)
Misc.	4 ch. RS-232C (D-sub 9 pin), I/O extensions (50 pin and 40 pin)

for example, it automatically changes the bobbin with turret. The control program was originally developed with Mitsubishi FX2N PLC, and then it was converted to VHDL by our converter.

It is practically difficult to make our converter fully compatible with FX2N PLC system, since it includes numerous instructions and peripheral devices. Therefore, we slightly reduced the optional features of the control program for the FPGA implementation, while sustaining the essential features. The omitted features include interactive user interface for parameter settings, automatic detection of fiber thickness, etc. This wider system successfully operated with FPGA control board to control various kinds of sensors and actuators.

Since we could not find any commercial FPGA board that fulfills our requirements, we designed and manufactured a new FPGA board for control applications in conjunction with Factory-Automation Electronics Inc. [23] Figure 10 displays the photographic image of our board, TUTFA-CYCII-FI, which consists of two parts; FPGA mezzanine card and I/O card. FPGA mezzanine card has an Altera Cyclone II FPGA [24] and power supply ICs on board. I/O card implements various I/O devices and connectors. The specifications of TUTFA-CYCII-FI are summarized in Table 4.

7. Discussion

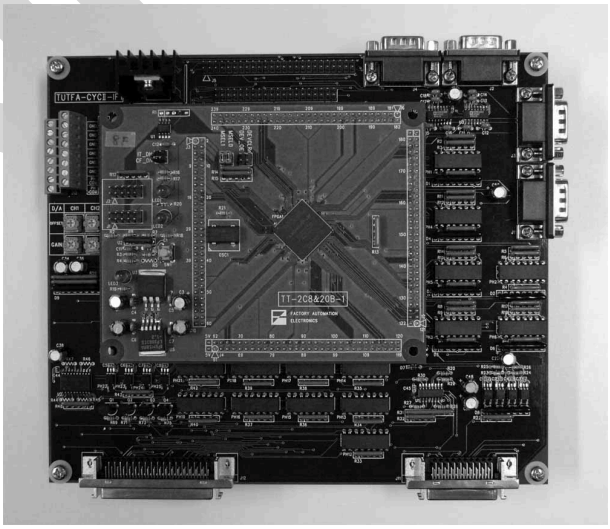


Fig. 10. TUTFA-CYCII-FI: an FPGA control board with an Altera Cyclone II device and various I/O channels.

7.1 Comments on Recent Publications

C. Economakos and G. Economakos [15] [16] followed our previous study [3], and added a few items to the advantages of FPGA implementation of PLC program. They evaluated control programs using C-based design tools, and verified the significant advantages of FPGAs in both fixed-point and floating-point applications. Their original contributions include (1) the quantitative evaluation of floating-point arithmetic operations, (2) the automatic exploration of the best design alternatives, and (3) various design options for higher performance.

Economakos's studies aim to automate the optimization of design, and apply various design options mechanically to examine possible options. The problem is that such strategy may generate many impractical designs, which does not satisfy the constraints imposed by I/O devices.

A simple example of this problem is the pipelining of circuits [15] [16]. Though the pipelined circuit may achieve higher

throughput, it has a larger latency than the original. A stream signal processing application might not be affected by this additional latency, while a feedback control application easily collapses by the additional latency.

In sequence control applications, it is generally essential to maintain the sequence and the timing constraints of I/O signals. The wrong sequence may lead to a malfunction or a mechanical breakdown to end up with a serious accident. These constraints are not explicitly given, but are implicitly implemented in the logic of PLC program. Thus, the only practical strategy is to preserve the sequence of I/O signals when a PLC program is converted into the corresponding circuit. If the sequence is not preserved, it is practically impossible to guarantee the proper control on the I/O devices.

Our tool was carefully designed to preserve the sequence of the original PLC program. Although our tool generates conservative designs, the behaviors of these designs are quite predictable and strictly preserve the order and context of inputs and outputs.

Despite all these problems, Economakos's works pointed out some interesting and productive topics. Economakos's works augment our work in many aspects, though they never replace ours.

7.2 Comments on Performance Advantage In Section 4, the performance advantages of our FPGA implementations were presented over PLC implementations. Although the advantages were outstanding, the evaluation was based on a single PLC product. The performance advantage might be offset by using other faster PLCs.

Mitsubishi Electric Corporation has two types of PLCs in their product line: MELSEC-Q series and MELSEC-F series. Our evaluation platform is FX2N, which belongs to F series and executes a basic operation (e.g., AND) in 80 ns [17]. Meanwhile, the current fastest PLC of Mitsubishi is Q26U in Q series, which executes an AND instruction in 9.5 ns [25]. According to the brochure [26], Q26U is the fastest PLC on the market in February 2009.

Roughly estimating, a Q26U PLC is expected to be 8–10 times faster than an FX2N PLC, while our Sequential (dedicated) and Flat designs are 184 and 8050 times faster than FX2N, respectively. Though the performance advantage might depend on the combination of FPGA device and PLC product, the advantage of FPGA implementation looks quite evident.

8. Conclusion

This study outlined a systematic method to implement a hard-wired sequence control from PLC software, which includes a converter that translates PLC instruction sequence into logic description, a control logic library to support various PLC instructions and peripheral devices, a design framework that integrates control logic and peripheral functions on an FPGA chip, and an experimental FPGA control board.

To show the advantages of our method, two sample ladder programs were examined and evaluated for Mitsubishi FX2N PLC and Altera Stratix II FPGA. Various logic designs of these ladder programs were investigated, and shown to fit into an off-the-shelf FPGA chip. The performance advantage over PLC technology was obvious. In case of a productive ladder program, Sequential design was estimated to be 184 times faster than PLC, and Flat design was 44 times faster than Sequential design (i.e., 8050 times faster than PLC).

A perfect layer winder system was actually built with our FPGA control board, whose logic design was generated by our design framework. This winder system successfully operated and exhibited the advantages of our scheme.

The following items are left for future study.

- More examples of control programs should be examined for a wide range of applications.
- The converter should be enhanced to support more control functions.
- The converter should be enhanced for more performance optimization.
- The converter should be enhanced to support more PLC platforms.
- Our design framework should be enhanced to work with embedded processors on FPGA devices.

Many old control systems are forced to abolition or reimplementation by the discontinuation of essential parts. Our framework might be useful for such situations.

Acknowledgment

The authors are grateful to Mr. Katsumi Asakura, the president of Yashima Netsugaku Co., Ltd. This work was partially supported by the Cooperation of Innovative Technology and Advanced Research in Evolutional Area (CITY AREA). Support for this work was also provided by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS).

(Manuscript received xxx , , revised xxx ,)

References

(1) Altera Corp., Stratix II Device Handbook, May 2007. http://www.altera.com/.

(2) IEEE standard VHDL language reference manual, 2002. IEEE Std 1076-2002.

(3) S. Ichikawa, M. Akinaka, R. Ikeda, and H. Yamamoto, "Converting PLC instruction sequence into logic circuit: A preliminary study," Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2006), pp.2930–2935, 2006.

(4) M.A. Adamski and J.L. Monteiro, "PLD implementation of logic controllers," Proc. IEEE Int'l Symp. Industrial Electronics (ISIE'95), pp.706–711, 1995.

(5) M. Adamski and J.L. Monteiro, "From interpreted Petri net specification to reprogrammable logic controller design," Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2000), pp.13–19, 2000.

(6) M. Wegrzyn, M.A. Adamski, and J.L. Monteiro, "The application of reconfigurable logic to controller design," Control Engineering Practice, vol.6, pp.879–887, 1998.

(7) A. Wegrzyn and M. Wegrzyn, "Petri net-based specification, analysis and synthesis of logic controllers," Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2000), pp.20–26, 2000.

(8) M. Ikeshita, Y. Takeda, H. Murakoshi, N. Funakubo, and I. Miyazawa, "An application of FPGA to high-speed programmable controller – development of the conversion program from SFC to Verilog –," Proc. 7th IEEE Int'l Conf. Emerging Technologies and Factory Automation (ETFA'99), pp.1386–1390, 1999.

(9) C.F. Silva, C. Quintans, J.M. Lago, and E. Mandado, "An integrated system for logic controller implementation using FPGAs," Proc. IEEE 32nd Annual Conf. Industrial Electronics (IECON 2006), pp.195–200, 2006.

(10) C.F. Silva, C. Quintans, E. Mandado, and M.A. Castro, "Methodology to implement logic controllers with both reconfigurable and programmable hardware," Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2007), pp.324–328, 2007.

(11) I. Miyazawa, T. Nagao, M. Fukagawa, Y. Ito, T. Mizuya, and T. Sekiguchi, "Implementation of ladder diagram for programmable controller using FPGA," Proc. 7th IEEE Int'l Conf. Emerging Technologies and Factory Au-

tomation (ETFA'99), pp.1381–1385, 1999.
(12) J.T. Welch and J. Carletta, "A direct mapping FPGA architecture for industrial process control applications," Proc. Int'l Conf. Computer Design (ICCD2000), pp.595–598, 2000.
(13) D. Du, Y. Liu, X. Guo, K. Yamazaki, and M. Fujishima, "Study on LD-VHDL conversion for FPGA-based PLC implementation," International Journal of Advanced Manufacturing Technology, vol.40, pp.1181–1190, 2009.
(14) Altera Corp., APEX 20K Programmable Logic Device Family Data Sheet, March 2004. DS-APEX20K-5.1.
(15) C. Economakos and G. Economakos, "FPGA implementation of PLC programs using automated high-level synthesis tools," Proc. IEEE Int'l Symp. Industrial Electronics (ISIE 2008), pp.1908–1913, 2008.
(16) C. Economakos and G. Economakos, "Optimized FPGA implementations of demanding PLC programs based on hardware high-level synthesis," Proc. IEEE Int'l Conf. Emerging Technologies and Factory Automation (ETFA 2008), pp.1002–1009, 2008.
(17) Mitsubishi Electric Corp., Programming Manual II: The FX series of programmable controller (FX1S/FX1N/FX2N/FX1NC/FX2NC), April 2003. JY992D88101 rev. D.
(18) Mitsubishi Electric Corp., "MELFANS web," 2009. http://wwwf2.mitsubishielectric.co.jp/melfansweb/english/.
(19) M. Chiang and R. Palkovic, "LCC simulators speed development of synchronous hardware," Computer Design, vol.25, no.5, pp.87–92, 1986.
(20) E.G. Coffman, ed., Computer and Job-shop Scheduling Theory, John Wiley & Sons, 1976.
(21) M.J. Gonzalez, "Deterministic processor scheduling," ACM Computing Surveys, vol.9, no.3, pp.173–204, 1977.
(22) "Yashima Netsugaku Co., Ltd.." http://www.yashima-ne.co.jp/.
(23) "Factory-Automation Electronics Inc.." http://www.fae.jp/.
(24) Altera Corp., Cyclone II Device Handbook, February 2008. http://www.altera.com/.
(25) Mitsubishi Electric Corp., QCPU Programming Manual (Common Instruction), July 2009. SH-080809ENG rev. C.
(26) Mitsubishi Electric Corp., Mitsubishi iQ Platform Programmable Controller MELSEC-Q Series [QnU], February 2009. L08101E rev. E.

Shuichi Ichikawa (Non-member) received his D.S. degree in Information Science from the University of Tokyo in 1991. He has been affiliated with Mitsubishi Electric Corporation (1991–1994), Nagoya University (1994–1996), and Toyohashi University of Technology (since 1997). Currently, he is an associate professor of the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. He is a senior member of IEEE and IEICE.

Masanori Akinaka (Non-member) received his M.E. degree in 2007 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Since April 2007, he is affiliated with Panasonic Semiconductor Systems and Technology Co., Ltd.

Hisashi Hata (Non-member) received his M.E. degree in 2009 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Since April 2009, he is affiliated with Renesas Technology Corporation.

Ryo Ikeda (Non-member) received his M.E. degree in 2005 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Since April 2005, he is affiliated with Toshiba IT & Control Systems Corporation.

Hiroshi Yamamoto (Non-member) received his M.E. degree in 2005 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Since April 2005, he is affiliated with Altech Corporation.