Evaluation of the hardwired sequence control system generated by high-level synthesis

Naoki Fujieda, Shuichi Ichikawa, Yoshiki Ishigaki, and Tasuku Tanaka
Department of Electrical and Electronic Information Engineering,
Toyohashi University of Technology,
Toyohashi, Aichi, Japan
`fujieda@ee.tut.ac.jp` (Naoki Fujieda),
`ichikawa@ieee.org` (Shuichi Ichikawa)

***Abstract*—This study presents the application of the commercial High Level Synthesis (HLS) to a hardwired control application with quantitative comparison to the traditional approach that uses logic synthesis with HDL. Though the derived circuits from HLS are comparable to that of logic synthesis, the design trade-offs in HLS are difficult to control. This study also presents the design and evaluation of the whole system of hardwired control with a Xilinx Zynq-7000 FPGA platform. From our experiments, two performance bottlenecks were identified: the RAM for memory elements that serializes the read/write accesses, and data transfer time via the peripheral bus. According to our results, the sole hardwired control was 10 times faster than the original software, while the overall performance was 4 to 50 times worse than the original software. The use of flipflops and dedicated I/O pins are necessary for high-performance systems.**

I. Introduction

Programmable Logic Controller (PLC) is widely adopted for the sequence control of industrial machinery. Although PLC is flexible and reliable, there are two problems; i.e., performance and security. The first problem is that the performance of PLC does not always satisfy the requirements of large or highly responsive control systems. The second problem comes from the fact that PLC software is an easy target to duplicate and analyze.

As an answer to these problems, there have been studies to implement PLC software as a hardwired control circuit on a reconfigurable logic device, e.g. FPGA (Field Programmable Gate Array). The performance of hardwired logic is generally higher than PLC software, and the flexibility of control logic can be sustained by the reconfiguration of an FPGA device. Hardwired control is also resistant to duplication and analysis, because the logic circuit is more difficult to target for analysis and duplication than software.

In early studies, Hardware Description Languages (HDL) and logic synthesis systems were used to generate a hardwired control circuit. Then, High Level Synthesis (HLS) systems appeared and became commercially available to generate a logic circuit from a popular programming language (e.g. C language). With HLS systems, designers can specify hardware functionality at higher level of abstraction to reduce the time and efforts required for hardware design [14].

On the other hand, HLS tends to be more complex in nature than logic synthesis. Many techniques and optimizations are applied to the design, which involves a large numbers of options and settings. Though the default features of HLS may yield good results in many cases, it is not always optimal for a specific application. It might be difficult for users to control HLS to generate a good hardwired control circuit, and the derived circuit might be larger or slower than the circuit generated from HDL with logic synthesis. All these points should be clarified before adopting HLS for practical projects.

The first purpose of this study is to examine the potential of commercial HLS for control applications, compared to the traditional approach that adopts logic synthesis and HDL. Three sample programs are converted to logic circuits with HLS and traditional methods, and the derived circuits are quantitatively evaluated and compared. Though various preceding studies reported the hardwired control generated by synthesis, HLS, and specially developed tools, they did not compare different approaches in a quantitative manner. To the best of the authors' knowledge, this is the first work that quantitatively discusses the pros and cons of the HLS approach over the traditional approach for hardwired control circuits.

Another purpose of this work is to examine the performance and the resource requirements of the whole system, which includes the hardwired control circuit as a peripheral device. Though the preceding studies focused on the hardwired control circuit itself (e.g., [5]), a practical system consists of many other parts such as an embedded processor, bus interfaces, memory modules, etc. The performance of the whole system might be much different from that of the sole hardwired control part, because various overheads are involved in the whole system. The resource requirement of the whole system should be also examined along with the performance. This study quantitatively discusses such practical aspects of the hardwired control.

The rest of this paper is organized as follows. Section II outlines the background and related studies of this work. Section III introduces the methods to convert PLC software into hardware. Section IV describes the overall design of hardware control system, and Section V presents the evaluation results. Section VI concludes the paper.

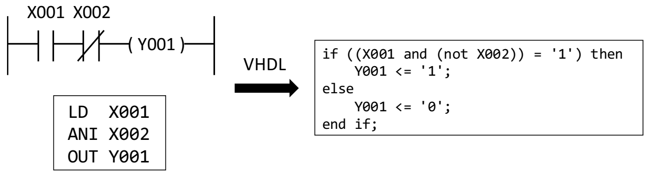
II. Related Studies

The hardware implementation of control logic has been studied since the middle of 1990s, along with the rapid evolution of FPGA technologies. Adamski [1] and Wegrzyn [15] presented systems that transfer the PLC program in Petri-net format into HDL. Ikeshita et al. [7] converted PLC software in SFC into Verilog HDL, while Miyazawa et al. [13] proposed to convert the ladder diagram to VHDL. These early studies adopted HDL and logic synthesis to generate the logic circuit that corresponds to the original PLC software.

Figure 1 illustrates an example of PLC software and the corresponding VHDL code [5]. First, a rung of ladder diagram is compiled into the corresponding PLC instructions. The converter then reads the PLC instructions and converts them into the corresponding VHDL, rung by rung. The condition part of the rung is converted into the corresponding conditional statement, and the output part is converted into the corresponding assignments.

The following studies explored the optimization schemes for hardwired control. As an example, Ichikawa et al. [5][6] proposed to convert PLC software into logic circuit for higher performance and higher security. They converted PLC instructions to VHDL code, which was then synthesized by using the commercial logic synthesizer. The performance advantages of three design options (Sequential Design, Levelized Design, and Flat Design) were quantitatively evaluated, where the parallelism in hardwired control was utilized. Du et al. [2] and Milik [9][8] also discussed the optimization techniques to generate a high-performance control circuit from a PLC instruction sequence.

Though the above studies discussed the techniques to utilize the parallelism in the PLC instruction sequence, many of these techniques are common and well known in the computer architecture or design automation communities. The optimization efforts might be drastically reduced by using HLS, which incorporates such optimization techniques [14]. To utilize recent HLS system, PLC instructions have to be converted into C language (Figure 2).



```
LD  X001
ANI X002
OUT Y001
```

```
if ((X001 and (not X002)) = '1') then
    Y001 <= '1';
else
    Y001 <= '0';
end if;
```

Fig. 1. An example of conversion from PLC program to VHDL [6].

Economakos [3][4] translated the STL language of Siemens S7 PLC into C language, which was then converted to HDL with Catapult C HLS software. Economakos discussed the effects of coding styles to the area and the performance in the FPGA implementations. The use of C-based HLS is particularly attractive when the user would like to implement the hardware partially, leaving most of the software executed as software. Recent HLS systems support hardware/software co-design, which enables users to examine the trade-offs between software and hardware. This approach is also expected to be applicable to C-based PLCs, which have become popular among PLC users.

Despite the pioneering works by Economakos, it is still indefinite whether the current HLS system can replace the old design scheme. If the optimizations of HLS are insufficient or not suited to hardwired control, the derived circuit might be slower or larger than that generated by HDL and logic synthesis. Even if HLS is well established, it might be difficult to derive the best result, since a commercial CAD system is a kind of black box. It applies every possible techniques in the black box, and shows the sole result of trade-offs out of vast variety of possibilities. It is sometimes difficult to control the optimization process, and to analyze the factors separately.

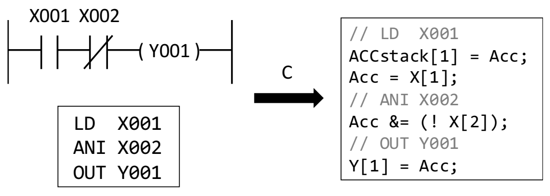
This work discusses the pros and cons of an HLS approach over the traditional approach for hardwired control circuits, using Xilinx tools and FPGAs.

III. Conversion of PLC program

Throughout this study, Mitsubishi MELSEC-Q series PLC [12] is assumed as the target platform. Mitsubishi GX Works [11] software with GX Converter [10] is adopted to generate the instruction list as a text file. The derived instruction list is specific to MELSEC PLCs, while it well resembles to the IEC-61131 instruction list. It is thus expected that the following discussion holds good in IEC-compliant PLCs.

A. PLC-to-C conversion

A custom-made converter was developed to convert a PLC instruction sequence into the corresponding code of standard C language. This converter is designated as the C-converter in the following discussion.



```
LD  X001
ANI X002
OUT Y001
```

```
// LD  X001
ACCstack[1] = Acc;
Acc = X[1];
// ANI X002
Acc &= (! X[2]);
// OUT Y001
Y[1] = Acc;
```

Fig. 2. An example of conversion from PLC program to C.

C-converter was designed to port a PLC instruction sequence to the corresponding embedded software written in C language. It was actually used in a project, and the derived C code successfully operated on a real-time OS. It should be noted that C-converter was designed without any consideration of hardware design.

C-converter converts each PLC instruction into the corresponding piece of C code. Figure 2 depicts the corresponding C code and the instruction list. A typical instruction set architecture (ISA) of PLC, including MELSEC-Q, is modeled as a stack machine. In Fig. 2, the stack is represented as the array ACCstack, while the stack top register is represented as the variable Acc. LD instruction pushes the previous result onto the stack, and loads its operand to the Acc register. The next ANI instruction applies AND operation to Acc with the inverted value of its operand. OUT instruction writes the value of Acc into the destination operand.

Though the relays (X and Y) are 1-bit wide, they were implemented by using *char* data-type (8-bit wide) in the current C-converter. This decision was made mainly for simplicity. If eight relays are packed into a char variable, the code would be more complicated and the performance would be degraded by additional shift and mask operations.

B. C-to-HDL conversion with Vivado HLS

The C code generated by C-converter is converted to HDL code by Xilinx Vivado HLS [17]. Vivado HLS automatically produces a micro-architecture that satisfies the desired performance and resource goals. For further optimization, users may designate various directives shown below.

- DATAFLOW directive enables task level pipelining, which enables the concurrent execution of functions and loops.
- UNROLL directive unrolls the loop by creating multiple independent operations instead of a single set of operations.
- PIPELINE directive enables the concurrent execution of operations within a loop or a function.
- ALLOCATION directive specifies a limit for the numbers of operations, cores or functions used.

These directives are described in the loops or functions to be optimized.

C. PLC-to-HDL conversion

To make a quantitative comparison between a C-based approach and the HDL-based approach, another converter was implemented to convert a MELSEC-Q instruction sequence into Verilog HDL descriptions. This converter was designed to reproduce the results of HDL-based design by Ichikawa et al. [5] Particularly, Sequential Design and Levelized Design are examined in the following sections. Flat Design was not implemented, because we could not find a way to reproduce its equivalent by a C-based approach. This converter is denoted as the HDL-converter in the following discussions.

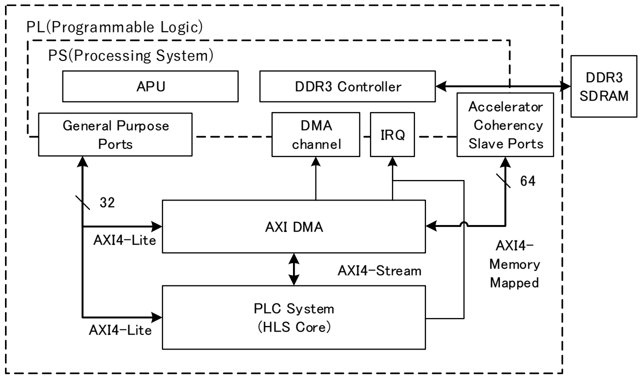


Fig. 3. Block diagram of the proposed system.

IV. SYSTEM DESIGN

A. Whole System

Figure 3 is the block diagram of our implementation, which consists of PLC system part that is the hardware generated by Vivado HLS from PLC instructions, AXI DMA IP core [16] that transfers data to/from the PLC system part, APU (Application Processor Unit) that executes software, and various interfaces that connect these parts. The PLC system part and DMA are controlled by APU via AXI4 Lite bus. Though the PLC system part is generated for each application, other parts are common and not affected by the target application.

The flow of operation is as follows. The PLC system part becomes idle and waits for data after initialization. APU executes the application software, and sends data to the PLC system part when it requires hardwired control. After sending data, APU waits for the finish flag of the PLC system part to be asserted. The PLC system part then begins processing the received data, and returns the data to APU via DMA when finished.

B. PLC System IP

The PLC system part is generated from the C code by HLS, where the C code includes the code generated by C-converter and the corresponding data transfer code. Figure 4 illustrates the block diagram of the PLC system part. In Fig. 4, AXI4-Lite interface, controller, BRAM, and its supplements are automatically generated by HLS. Since the AXI4-Stream is 16-bit wide, two char-type data are packed in each transfer. Though the relay data is represented by using char-type variables as stated earlier, the hardware actually requires 1-bit data out of an 8-bit char. Data Converter deals with such data-type conversion and packing/unpacking.

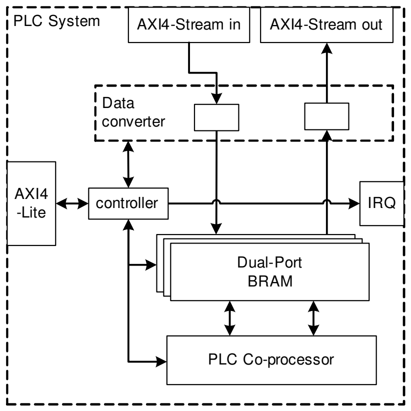


Fig. 4. Block diagram of the PLC System IP generated by HLS.

The flow of operation is summarized as follows. After initialization, APU sets the Start flag of the PLC system part via AXI4-Lite. PLC system part becomes idle and waits for the incoming data. The input data are received from the AXI4-Stream, processed by the Data Converter, and then written into BRAM. The PLC system part starts processing when all data are stored in BRAM. Then, the PLC system part goes back to idle state after finishing processing and sending back the results via the AXI4-Stream.

V. Evaluation

A. Methodology

The whole system, including the PLC Co-processor, was implemented with Xilinx FPGA to evaluate resource usage and processing time. The differences between this implementation and the previous study [5] are summarized as follows. The data are stored in BRAM, which are accessed by the APU and PLC Co-processor in this work, while the data were stored in flipflops (FFs) in the previous study. Multiplication and division are processed in multiple cycles in this work, while a single-cycle multiplier and divider were adopted in the previous study.

When using C-converter, all internals of the PLC system part are generated from C codes. Meanwhile, for the HDL-converter, the components of the PLC system part have to be described in Verilog HDL as individual modules, e.g. PLC Co-processor, AXI4-Stream interface, BRAM, and other controllers in Figure 4. In using C-converter, BRAM is automatically scheduled by HLS. On the other hand, when HDL-converter is used, one port of BRAM is connected to the AXI4-Stream and the other port is connected to the PLC Co-processor; each port is scheduled independently.

For the evaluation, Digilent ZedBoard (Rev. D) with Xilinx Zynq XC7Z020-CLG484 was adopted. Xilinx Vivado, Vivado HLS, and SDK (ver. 2016.3) were used for development. All options for synthesis and implementation were set to default. The clock frequency of PS (Processing System) was 666.7 MHz, while one of three standard clocks (1333, 1067, 1000 MHz) was used for PL (Programmable Logic) with an appropriate divider setting.

Three PLC applications are implemented and evaluated as shown below. PID is a PID controller (21 instructions), which is identical to PID in Ichikawa et al. [5]. YNK is a sample control program (165 instructions), also used in [5]. Plant is a part of practical control program (3755 instructions). Since the previous study reported the effect of resource binding, this work also investigates the ALLOCATION directive to limit the number of resources.

To examine the resource usage, the numbers of LUT, FF, DSP, and BRAM are measured. The maximal operational frequency f_{max} is also measured, where f_{max} is calculated from the worst negative slack (WNS) at the target frequency f_{targ}. In the evaluation of the whole system, the average execution time was measured with the performance counter of PS (10.42 Mtick/s) by invoking PLC system IP 10000 times. In the evaluation of the sole PLC Co-processor, the worst execution cycles were estimated by HLS.

B. Results of PLC Co-processor

Table I lists the evaluation results of PLC Co-processor (stand-alone). Roughly speaking, LUT corresponds to the combinational gates, FF corresponds to registers, DSP corresponds to multipliers, and BRAM corresponds to internal SRAM.

The uppermost group shows the results derived with C-converter and HLS. C-converter was designed without any considerations for hardware design, but the derived C code was successfully synthesized with the least modification such as header files. The middle and lowermost groups shows the results of Sequential Design and Levelized Design, derived with the HDL-converter and logic synthesis. PID-bind represents the results of PID with resource constraint, where the numbers of multiplier and divider are limited to one.

From Table I, it is evident that the resource constraint leads to the reduction of hardware resources. In Sequential Design without resource constraint, the number of multiplier is equal to the number of multiply instructions [5]. The respective reduction of DSP is 1/3 (PID) and 1/12 (YNK), which is rational considering the numbers of multiply instructions: 3 (PID) and 12 (YNK). Though DSP of Plant was not reduced by resource constraint, this behavior was caused by the current specification of the HDL-converter. Plant includes 17 multiply instructions, all of which are 16-bit multiplication. Since a 16-bit multiplier is very small, the current HDL-converter does not share the multiplier and generates multipliers for each multiply instructions. (This behavior is reasonable, but might be regarded as a bug for this experiment.)

The input multiplexers are required for shared multipliers and dividers, which leads to the increase of LUTs. On the other hand, LUT will be reduced by sharing dividers, because dividers are implemented with LUTs. In YNK and Plant, LUT was actually reduced by sharing dividers, where the numbers of divide instructions are 9 for YNK and 66 for Plant. Meanwhile, PID includes no divide instruction, i.e. no dividers

TABLE I
EVALUATION RESULTS OF THE PLC CO-PROCESSOR.

C-converter with Vivado HLS									
	LUT	FF	DSP	36kb BRAM	f_{targ} [MHz]	WNS [ns]	f_{max} [MHz]	#cycle	Est.Time [us]
PID	385	719	12	0	318	0.047	322.8	19	0.06
PID-bind	395	501	4	0	350	0.038	354.7	19	0.05
YNK	4,711	4,379	44	0	239	0.089	244.2	56	0.23
YNK-bind	3,668	2,875	4	0	227	0.069	230.6	56	0.24
Plant	39,996	34,992	59	0	170	0.210	176.3	2,104	11.9
Plant-bind	28,747	23,877	4	0	174	0.072	176.2	2,319	13.2
HDL-converter (Sequential Design)									
	LUT	FF	DSP	36kb BRAM	f_{targ} [MHz]	WNS [ns]	f_{max} [MHz]	#cycle	Est.Time [us]
PID	575	981	12	0	242	0.165	252.1	55	0.22
PID-bind	676	726	4	0	239	0.014	239.8	55	0.23
YNK	4,215	3,803	48	0.5	206	0.018	206.8	473	2.29
YNK-bind	2,942	1,999	4	0.5	168	0.080	170.3	473	2.78
Plant	28,122	24,357	14	5.0	84	0.088	84.6	2,333	27.6
Plant-bind	23,481	22,258	14	5.0	88	0.145	89.1	2,333	26.2
HDL-converter (Levelized Design)									
	LUT	FF	DSP	36kb BRAM	f_{targ} [MHz]	WNS [ns]	f_{max} [MHz]	#cycle	Est.Time [us]
PID	577	983	12	0	242	0.130	249.9	55	0.22
PID-bind	685	725	4	0	245	0.093	250.7	55	0.22
YNK	4,015	3,753	48	0.5	233	0.000	233.0	241	1.03
YNK-bind	2,807	1,954	4	0.5	200	0.036	201.5	357	1.77
Plant	27,140	24,264	14	3.5	114	0.154	116.0	1,986	17.1
Plant-bind	23,607	22,189	14	3.5	126	0.152	128.5	1,986	15.5

TABLE II
COMPARISON OF EXECUTION TIME [US].

	Ichikawa et al. [6]			HDL-converter of this work		
	Sequential	Levelized	S/D ratio	Sequential	Levelized	S/D ratio
PID	0.111	0.0793	1.40	0.22	0.22	1.00
PID-bind	0.137	0.102	1.34	0.23	0.22	1.05
YNK	8.74	1.50	5.83	2.29	1.03	2.22
YNK-bind	11.4	2.50	4.56	2.78	1.77	1.57

to be reduced. Thus, the LUT of PID increased by resource constraint for the input multiplexers of multipliers, in exchange for the reduction of DSP.

Table II summarizes the execution times of PID and YNK for Sequential and Levelized designs with/without resource constraint. It should be noted that FPGA platforms used are different between this work (Xilinx Zynq-7000) and previous work (Altera Stratix-II), and that the absolute values should not be compared. The ratio of Sequential to Levelized (S/D ratio) is considered to represent the degree of parallelism in the application, which should be comparable in these two studies. However, Table II suggests that the S/D ratio of this study is lower than the previous results. It is caused by the difference of the memory elements. As stated in Section V-A, the memory elements of PLC program are implemented by using BRAM in this study, while the previous study assumed flipflops. Though flipflops can be used physically in parallel, the accesses to BRAM are serialized. This substantially reduces the parallelism in Levelized Design of this work. Considering this difference, it is regarded that HDL-converter of this work reasonably reproduced the previous results.

For all three applications, the new scheme (C-converter + HLS) generated faster circuits than the old scheme (HDL-converter + synthesis). These results confirmed that the recent HLS can generate the hardwired control circuit of the reasonable performance from C code. As for resource requirement, the new scheme yields comparable results to the old scheme, though the inclinations are different. In PID, the new scheme is better and yields a smaller and faster circuit. In YNK and Plant, the new scheme tends to consume larger resources with larger performance. With HDL-converter, BRAMs were inferred, which leads to the reduction of LUT and FF with potential performance bottleneck by long wire delays.

In short, the resource requirements of the two schemes are regarded comparable. The new scheme yielded a performance-oriented circuit, while the old scheme yielded a cost-oriented circuit. This is a result of trade-off, and HLS may generate a smaller circuit if the resource constraints are severe. This aspect might be observed in the next section, which presents the evaluation results of the whole system.

TABLE III
EVALUATION RESULTS OF THE WHOLE SYSTEM.

C-converter with Vivado HLS

	LUT	FF	DSP	36kb BRAM	f_{targ} [MHz]	WNS [ns]	f_{max} [MHz]	Time [us]
PID	3,767	4,910	12	5.5	152.4	0.368	161.5	68.9
PID-bind	3,813	4,689	4	5.5	152.4	0.160	156.2	68.9
YNK	5,962	7,125	44	8	125.0	0.318	130.2	146.0
YNK-bind	5,735	5,898	4	8	90.9	0.255	93.1	145.9
Plant	18,510	13,749	44	12	58.0	0.658	60.3	671.9
Plant-bind	15,930	12,050	4	12	76.2	0.093	76.7	673.3

HDL-converter (Sequential Design)

	LUT	FF	DSP	36kb BRAM	f_{targ} [MHz]	WNS [ns]	f_{max} [MHz]	Time [us]
PID	3,965	5,178	12	11	152.4	0.357	161.1	18.4
PID-bind	4,068	4,922	4	11	152.4	0.027	162.7	18.4
YNK	7,429	8,017	48	11.5	152.4	0.358	161.2	146.0
YNK-bind	6,422	6,214	4	11.5	166.7	0.132	170.4	146.0
Plant	31,318	28,563	14	16.0	82.1	0.094	82.7	547.4
Plant-bind	27,679	26,477	14	16.0	83.3	0.029	84.1	539.4

HDL-converter (Levelized Design)

	LUT	FF	DSP	36kb BRAM	f_{targ} [MHz]	WNS [ns]	f_{max} [MHz]	Time [us]
PID	3,975	5,180	12	11	166.7	0.126	170.2	18.0
PID-bind	4,084	4,923	4	11	166.7	0.027	167.4	18.0
YNK	7,288	7,969	48	11.5	166.7	0.161	171.3	133.8
YNK-bind	6,308	6,176	4	11.5	166.7	0.051	168.1	134.6
Plant	30,486	28,482	14	14.5	125.0	0.067	126.0	363.5
Plant-bind	26,751	26,403	14	14.5	118.5	0.058	119.3	361.9

C. Results of Whole System

Table III summarizes the evaluation results of the whole system. In this table, the column *Time* stands for the average time to call PLC system IP from APU.

With the old scheme, the resource requirements increased for the additional units, where the approximate increases of LUT, FF, and BRAM were 3400, 4200, and 11 respectively. Max operational frequency (f_{max}) was reduced to 160–170 MHz, which might be caused by AXI DMA IP core whose maximum frequency is 180 MHz [16].

With the new scheme, PID becomes slightly smaller but 3.8 times slower than the old scheme. YNK becomes also smaller, and the execution time is almost the same. In the case of PID, the new scheme generates 40–50 % smaller circuits with 1.8 times longer execution time than the old scheme. Longer execution time was a consequence of the low f_{max}, which was caused by the long critical path to BRAM in the Data converter. Instead, the other parts of the system were highly optimized with the low f_{max}, resulting in a much smaller circuit than the old scheme. These results suggest that HLS tends to generate an area-optimized circuit for a large design, while generating a performance-oriented circuit for a small design. This behavior is reasonable, but the users might be frustrated when they would like to make a different choice intentionally.

The overhead to call PLC system IP is 1.1 to 3.8 times larger in the new scheme. From Tables I and III, it is clear that more than 95% of the execution time in Table III is the calling overhead, which is mostly DMA transfer via AXI. If the C codes of PID, YNK, and Plant are executed on PS, their execution times are 1.3, 5.2, and 165.7 [us], respectively.

In short, the PLC system IP itself is approximately 10 times faster than PS, while the performance of the whole system is 4 to 50 times lower than PS. The performance advantage of hardwired logic was totally offset by the bus transfer overhead in this case. To avoid such a situation, the hardwired logic should be more autonomous. The data transfer via peripheral bus should be reduced, and the hardwired control should have the dedicated I/O pins for control applications.

VI. CONCLUSION

This work examined the potential of commercial HLS for hardwired control applications. Compared to the old scheme with HDL and logic synthesis, the new scheme with C and HLS generated almost comparable circuits considering area/performance trade-off. Although the automatic trade-off in the new scheme works reasonably, users may be frustrated or bewildered to control HLS to generate an intended circuit significantly different from the default of HLS.

This work also constructed the whole system that includes the hardwired control circuit as a peripheral device. In the evaluations, we found two performance bottlenecks. The first was the memory module (BRAM), which serialized the read/write accesses. The second was the data transfer via peripheral bus. For high-performance control circuits, flipflops are required instead of RAM, and the dedicated I/O pins are necessary.

The authors are going to apply the presented system to protect the intellectual property and the security of control systems. In this case, the performance overhead is not a serious problem. Instead, some methods to increase the complexity of analysis will be required.

Acknowledgement

This study was partially supported by JSPS Grants-in-Aid for Scientific Research (KAKENHI), Grant Number 26870278 and 16K00072.

References

[1] M. A. Adamski and J. L. Monteiro, “PLD implementation of logic controllers,” in *Proc. IEEE Int’l Symp. Industrial Electronics (ISIE’95)*, vol. 2, 1995, pp. 706–711.

[2] D. Du, X. Xu, and K. Yamazaki, “A study on the generation of silicon-based hardware PLC by means of the direct conversion of the ladder diagram to circuit design language,” *International Journal of Advanced Manufacturing Technology*, vol. 49, no. 5, pp. 615–626, 2010.

[3] C. Economakos and G. Economakos, “C-based PLC to FPGA translation and implementation: The effects of coding styles,” in *Proc. 16th Int’l Conf. System Theory, Control and Computing*, 2012, pp. 1–6.

[4] ——, “Efficient High-Level Coding in a PLC to FPGA Translation and Implementation Flow,” *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*, pp. 269–276, 2015.

[5] S. Ichikawa *et al.*, “Converting plc instruction sequence into logic circuit: A preliminary study,” in *Proc. 2006 IEEE Int’l Symp. Industrial Electronics (ISIE 2006)*, vol. 4, 2006, pp. 2930–2935.

[6] ——, “An FPGA implementation of hard-wired sequence control system based on PLC software,” *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 6, no. 4, pp. 367–375, 2011.

[7] M. Ikeshita *et al.*, “An application of FPGA to high-speed programmable controller – development of the conversion program from SFC to Verilog –,” in *Proc. 7th IEEE Int’l Conf. Emerging Technologies and Factory Automation (ETFA’99)*, vol. 2, 1999, pp. 1386–1390.

[8] A. Milik, “On Hardware Synthesis and Implementation of PLC Programs in FPGAs,” *Microprocess. Microsyst.*, vol. 44, no. C, pp. 2–16, Jul. 2016.

[9] A. Milik and E. Hrynkiewicz, “Synthesis and Implementation of Reconfigurable PLC on FPGA Platform,” *International Journal of Electronics and Telecommunications*, vol. 58, no. 1, pp. 85–94, 2012.

[10] Mitsubishi Electric Corp., *GX Converter Version 1 Operating Manual*, 2010, technical Manual IB(NA)-0800004-J.

[11] ——, *GX Works2 Version 1 Operating Manual (Common)*, 2016, technical Manual SH(NA)-080779ENG-AD.

[12] ——, *MELSEC-Q/L Programming Manual (Common Instruction)*, 2017, technical Manual SH(NA)-080809ENG-T.

[13] I. Miyazawa *et al.*, “Implementation of ladder diagram for programmable controller using FPGA,” in *Proc. 7th IEEE Int’l Conf. Emerging Technologies and Factory Automation (ETFA’99)*, vol. 2, 1999, pp. 1381–1385.

[14] R. Nane *et al.*, “A survey and evaluation of FPGA high-level synthesis tools,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[15] M. Wegrzyn *et al.*, “The application of reconfigurable logic to controller design,” *Control Engineering Practice*, vol. 6, pp. 879–887, 1998.

[16] Xilinx, *LogiCORE IP AXI DMA v7.1*, Oct. 2016, PG021.

[17] Xilinx, *Vivado Design Suite User Guide: High-Level Synthesis*, Nov. 2016, UG902 (v2016.4).