Transforming Ladder Logic to Verilog for FPGA Realization of Programmable Logic Controllers

Giancarlo Corti
Department Mechanical and
Manufacturing Engineering
Miami University
Oxford, Ohio 45056
Email: corticlg@miamioh.edu

Drake Brunner, Naoki Mizuno, and Peter Jamieson
Department of Electrical and
Computer Engineering
Miami University
Oxford, Ohio 45056
Email: jamiespa@miamioh.edu

***Abstract*—Programmable Logic Controllers (PLCs) are used in many industrial settings to control and automate machinery in a manufacturing process. Typically, these devices are programmed in ladder logic, which is used to define the logical control of connected machines in parallel. The resulting system needs to control machines in the millisecond time domain, and therefore, PLCs can implement what appears to be millisecond parallel control by emulating the logic with GHz processing capabilities of modern processors. This system solution works, but PLCs are expensive and cannot support all design implementations, and our work begins to support a community examination to replace the control computation resources with FPGAs. The reason for this shift in technology is FPGAs are by nature parallel, programmable (reconfigurable), low cost, and have a high pin capacity, which makes them excellent substitutes for PLCs. To evaluate this, however, the first step is to build a tool that converts ladder logic to a format mappable to FPGAs. For this, we have created an open-source tool called, Hashigo, that converts ladder logic to synthesizable Verilog. In this work, the tool is used to convert a ladder logic design to Verilog that is then mapped to an FPGA, and we verify that the resulting FPGA control is equivalent to that of the same benchmark implemented on an existing commercial PLC. For the benefit of the community, this software is released to the community as open-source so that both academic and commercial possibilities in this technology can be further advanced.**

I. Introduction

Programmable Logic Controllers (PLCs) were introduced to the manufacturing industry to replace large relay circuits used as control devices. This adds a huge benefit of flexibility to industrial control since the control can be more easily changed. Programmable controllers are, typically, connected to industrial equipment such as assembly lines and machine tools to sequentially operate the system in accordance with a stored control program [1]. PLCs implement the parallel logic analysis via processors working at GHz speeds to emulate microsecond logic operations, and these devices are used since they can be reprogrammed instead of rewired or adding new complex control circuitry to an existing design.

To make the transition from relays to PLCs intuitive to engineers who designed and built the relay control circuitry, a programming language called ladder logic was created. This language is essentially a circuit diagram of the equivalent relay logic. Ladder logic has been a popular programming language for PLCs ever since.

There are, however, fundamental differences between relay circuits and PLCs. Relay circuits operate under the control of combinational logic and execute in parallel with other relay circuits. Each relay functions on its own, changing states as fast as changes occur at the inputs. PLCs, on the other hand, use a sequential processor that only completes roughly an operation per clock cycle, and therefore the PLC needs to emulate the relay circuitry that it replaces, and does so at faster speeds than the relays so that it appears that the control is done in parallel. This is a common approach similar to that of video games making games appear to be animated, but merely presenting static pictures at a rate that humans cannot perceive the difference between the static images.

This can create some challenges for performance as the complexity of the control increases. To emulate relay logic, the PLC runs a repetitive emulation cycle that includes reading inputs, executing logic, and the updating outputs. The logic is scanned rung by rung (where a rung is a logical calculation) and coil by coil to determine the output. The cyclical nature means that the PLC is not performing real-time control, and as both the number of rungs and complexity of each rung increases the computational load on the PLC increases. Additionally, PLCs are expensive systems to purchase including the software support and service for these devices.

The reality of ladder logic is that when looked at from a digital system perspective is that basically the ladder logic is a set of logical statements. This means that the FPGA can implement the logic directly on its programmable logic fabric such that the processing can be done in real-time similar to the original relay circuits. Not only can FPGAs better handle the increasing complexity of control designs, but these devices are relatively inexpensive when compared to PLCs and have a high pin count to handle the input/output demand. The limiting factor with FPGAs is that the engineers designing the control are far from being familiar with typical FPGA hardware design using either schematic design or Hardware Description Language (HDL) design. The expertise needed to create their algorithms on an FPGA is not worth their time.

For this reason, our work presents and contributes an open-source compilation tool called, Hashigo, that takes ladder logic and converts it into synthesizable Verilog. This conversion allows the converted design to be mapped into a typical FPGA CAD flow that then creates a programmable file that can be mapped to an FPGA. With our tool we show how a ladder logic benchmark can be mapped to both a commercial

PLC and an FPGA prototype board. This demonstrates the tools correctness and also demonstrates how a cheap FPGA prototyping board can easily handle a real-time load of a ladder logic benchmark. Additionally, we release this tool as open-source so that other researchers and commercial entities can advance the possibilities for this implementation.

The remainder of this paper is organized as follows: Section II describes the existing work in this domain. Section III describes how Hashigo is implemented, and Section IV describes our benchmark and its implementation on both an FPGA prototyping board and a commercial PLC. Finally, V concludes the paper and describes future directions.

II. Background

This work is not the first foray into PLC-like implementation on FPGAs. A number of researchers have and are still working on how reconfigurable devices can be used to implement ladder logic.

The first attempt at implementing control on Programmable Logic Devices (PLDs) was by Adamski and Monteiro [2] where they viewed the control problem from the perspective of petri nets. In their work, the goal is purely to implement control on a PLD and not to convert existing PLC designs over to a PLD. They followed up this work with a later publication [3] that improved on their technique. Also, this work was further extended with Wegrzyn *et al.* [4] where they used the petri net descriptions, but now mapped their design to VHDL [5] so that they could target commercial CAD flows which map to devices such as FPGAs and Application Specific Integrated Chips (ASICs).

Soon after Petko and Karpiel [6] developed another technique to create control algorithms for FPGAs and ASICs and map them to simulink so that their complete system environment could be simulated both electrically and mechanically. Their methodology is not a completely autonomous flow, but they provide a number of steps, both automated and manually done, to map their control designs to a silicon implementation in either target language - Verilog or VHDL. The input to their system, however, is not something like ladder logic, and instead, the complete control is designed by an engineer familiar with their flow.

More recently, Economakos and Economakos [7] [8] developed a fully automated flow that takes in a PLC language input (Siemens Statement List programming language of the well-known Simatic S7-300/400 PLCs) and converts a design to a C implementation. With the C language implementation they then input the design into the Catapult C tool flow created by Mentor Graphics that can target FPGAs. Their main motivation is to efficiently implement more complex control designs that may not be feasibly implemented on a PLC.

Du *et al.* work [9] is the most similar to this work in that they create a tool to map ladder logic to VHDL. Their motivation is to improve performance by using an FPGA and they target Xilinx FPGAs, specifically. The limitation with their work is the tool is not open-source and so there has been no continuance or commercialization of their original work.

Finally, Milik [10] has created a flow from IEC61131-3 ladder logic design to FPGAs via Verilog as the CAD flow design entry language. Millik spends significant time on optimizations in the conversion using compile-time approaches such as building a directed flow graph and analyzing these graphs. This, arguably, is the most advanced tool in this research progression, but the tool is not available to the public, and we believe this is a crucial step if FPGAs are ever to become a competitive technology in manufacturing design in competition with PLCs.

In addition to control designs mapped to FPGA, this work is built with similar ideas and philosophies for open-source tool flows in the FPGA domain such as Icarus [11] and [12], which are open source tools for Verilog compilation for both emulation and synthesis, respectively. Tools such as VTR [13] and [14] are other open-source FPGA CAD tools that have helped our communities due to their easy and open access.

III. Hashigo

Hashigo is an open-source compiler that converts ladder logic designs, typically, designed and compiled for PLCs, into synthesizable Verilog that can target any silicon-based CAD flow (such as FPGAs and ASICs). In particular, the intended target for this tool is an FPGA CAD flow since this implementation makes the most sense due to the FPGAs reprogrammable characteristic. In this section, we will describe the details of this tool including some of the technologies that were used.

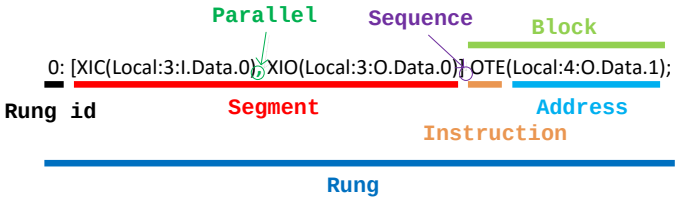

Fig. 1. A language breakdown of the HIL for Hashigo

Hashigo (which means "ladder" in Japanese) is built as a typical source to source compiler. The input language is ladder logic in an XML readable format. Specifically, our flow starts with an export of the ladder logic file from AllenBradley's RSLogix 5000 IDE to an XML file. Using the pugixml (http://pugixml.org/) XML parsing library we first convert the XML design into an intermediate language called Hashigo Intermediate Language (HIL) where files have "hshg" suffix to identify these. The reason for including this intermediate step instead of just parsing the XML directly is that this allows for other industrial tool flows to convert their designs, whether from XML or another output format, into the HIL format to be compatible with our tool. Figure 1 shows some of the language breakdown for the HIL where a rung of ladder logic is a single statement that internally has instructions and ordering that defines the logic, instructions, and naming.

The HIL format is then read into Hashigo. This conversion step is a typical compilation process in C/C++ where the Bison [15] and Flex [16] tools are used to parse the HIL grammar into an Abstract Syntax Tree (AST) and symbol table. Once the design is parsed into these two data structures the tool can perform any optimizations or transformations, but at present we do not consider any optimizations since we want to keep the tool both simple and operational for other researchers as their starting framework.

To generate the synthesizable Verilog, we traverse the AST and output the Verilog. Since ladder logic is relatively simple in terms of being logical descriptions the Verilog design is not very complex and is simply a finite state machine which implements the logic for an emulation cycle of the control. The

most complex components in ladder logic designs are timers that can be easily implemented in sequential logic. The only additional parameter needed to handle timers is the tool needs knowledge of the synthesized clock speed to adjust timing steps.

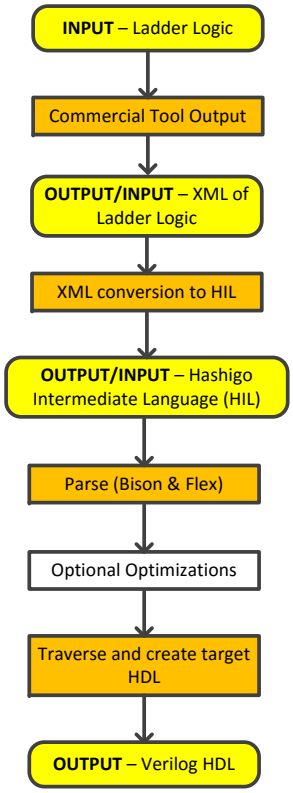

Fig. 2. A flow diagram of the Hashigo steps and input/outputs

Figure 2 summarizes the above steps where yellow boxes are for inputs and outputs, orange boxes are for steps currently implemented in Hashigo in the open-source release, and a clear box is where optional optimizations could be performed. Our code is written in C++ and the needed libraries for compilation are typical compiler libraries including Bison and Flex as described above.

Figure 3 shows an expanded flow in Figure 2 except the inputs and outputs now show an example design progressing in each of the yellow boxes. First, we start with a simple two rung ladder logic diagram. Next, we show a sample of how the XML looks like for this sample. In the next two boxes, the HIL and a Verilog design are provided as the design progresses through the flow. Though this is not an overly complex example, the example shows how the data is processed through the tool.

IV. Benchmark Verification

The goal of this work is to create an open-source tool that converts ladder logic to synthesizable Verilog. This section describes the benchmark we have created to both verify that our tool works and test the features supported by our tool.

The benchmark we provide with the tool is a car wash control circuit that includes 10 rungs of ladder logic and includes 3 timers and 1 loop control state. First, we implemented this benchmark in ladder logic in RSLogix 5000 IDE and then mapped it to execute on the Control Logix5000 PLC to observe both the functionality and verify that our benchmark worked. We chose this tool flow because we have both the software and hardware in house, but our design approach attempts to be general enough so that any industrial flow could be supported with some additional implementation to get the ladder logic (or similar design) into our HIL format as described earlier.

TABLE I. Resource Usage of the Car Wash Benchmark on a Cyclone IV FPGA

Resource	Report
Logic Elements	267 Logic Elements (<1% of FPGA)
Clock Speed	156.57 MHz

Next, we used our tool-flow to create synthesizable Verilog once the RSLogix 5000 IDE outputs the design as an XML file. Progressing through the flow, as described in the previous section, our final output is a Verilog file that implements the benchmark. We include this design as part of a design in Altera's (Intel) FPGA flow in Quartus 13.1 [17] targeting DE2-115 FPGA prototyping boards. These boards are created and manufactured by Terasic and include a Cyclone IV FPGA clocked at 50MHz. Table I shows both the area consumption and operating frequency of our design when mapped through this flow. Note, as stated earlier ladder logic control is not necessarily overly complex, and a Cyclone (which is a low-end FPGA in terms of cost, size, and performance) can easily implement this benchmark. This is not to say that more complex designs will always map to an FPGA implementation at the required speed, but rather, the idea of using FPGAs is highly feasible option that may be more efficient in terms of both cost and performance.

Finally, we program the design to the FPGA and verify that the operation of the benchmark matches that of the PLC to demonstrate that our implementation is correct.

V. Discussion and Conclusion

In this work, we presented, Hashigo, a tool to convert ladder logic to synthesizable Verilog with the intention of mapping the control to FPGAs as a replacement for PLCs. The main purpose of this work is to provide commercial interests and researchers with an open-source tool that can map ladder logic (in theory from any tool flow with some modification) to an implementation on an FPGA. We described the design of our tool, some of the choices we made, and implementation details. Finally, we verified that our tool works by mapping a benchmark to both an industrial PLC and an FPGA, and observed that the control sequence was equivalent.

The immediate next step in this work is to implement an FPGA controlled system that uses our design tool into an automated device. This involves controlling external power supplies via the FPGA outputs to activate transistors or relays that then are connected to actuators. From the input perspective we also need to interface the FPGA with sensors to allow for both input and feedback signals from the system.

An alternative future direction would be to support all PLC input languages such as ladder logic, structured text, functional block diagram, and sequential function chart and all the functionality associated with each language. However, functionality and syntax varies among each brand of PLC and this would be a major challenge that might be best supported as a community as opposed to individual efforts.

Another aspect that can be improved is regarding optimizations that can be made to designs under certain scenarios

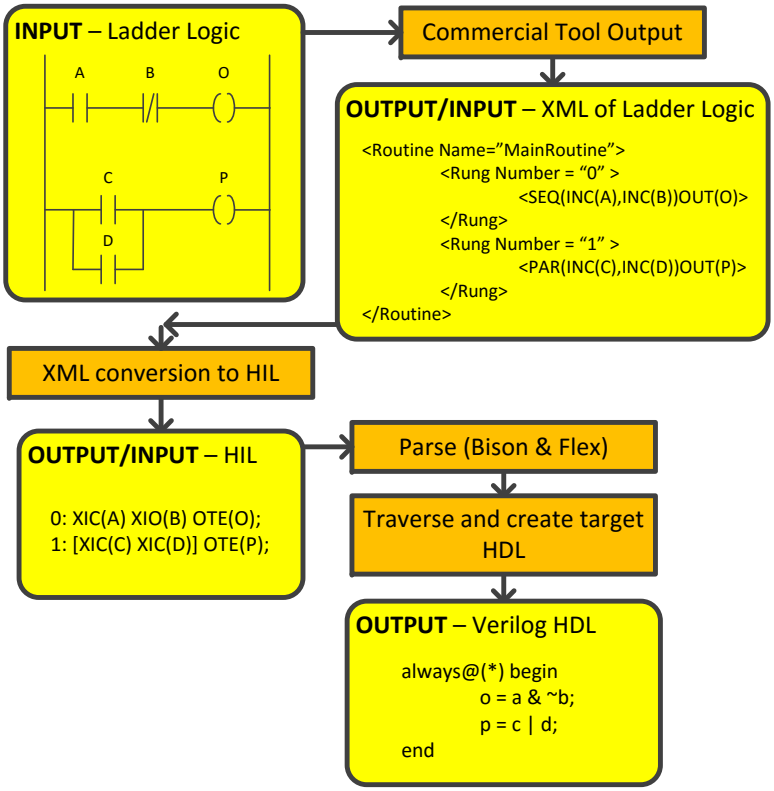

Fig. 3. The flow with example inputs and outputs for the stages

including the range of compiler optimizations that exist. One of these scenarios, for example, happens when there are no latches in the control design. This occurs when there are no data dependencies in the ladder logic program, and this implies that a purely combinational design can be implemented in the Verilog. This ensures that the functionalities from ladder logic are preserved while making the design scalable, since the required clock speed does not depend on the number of rungs but rather the delay of the longest rung as implemented by logic.

As one of the key deliverables with this work is the tool itself, researchers are welcome to download and use/modify our software from https://github.com/NigoroJr/hashigo.

References

[1] E. Dummermuth, "Programmable logic controller," 1976, uS Patent 3,942,158.

[2] M. A. Adamski and J. L. Monteiro, "Pld implementation of logic controllers," in *Industrial Electronics, 1995. ISIE'95., Proceedings of the IEEE International Symposium on*, vol. 2. IEEE, 1995, pp. 706–711.

[3] M. Adamski and J. L. Monteiro, "From interpreted petri net specification to reprogrammable logic controller design," in *Industrial Electronics, 2000. ISIE 2000. Proceedings of the 2000 IEEE International Symposium on*, vol. 1. IEEE, 2000, pp. 13–19.

[4] M. Wegrzyn, M. A. Adamski, and J. L. Monteiro, "The application of reconfigurable logic to controller design," *Control Engineering Practice*, vol. 6, no. 7, pp. 879–887, 1998.

[5] *IEEE Standard VHDL Language Reference Manual*, IEEE, 1987.

[6] M. Petko and G. Karpiel, "Semi-automatic implementation of control algorithms in asic/fpga." in *ETFA (1)*, 2003, pp. 427–433.

[7] C. Economakos and G. Economakos, "Fpga implementation of plc programs using automated high-level synthesis tools," in *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1908–1913.

[8] ——, "Optimized fpga implementations of demanding plc programs based on hardware high-level synthesis," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1002–1009.

[9] 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," *The International Journal of Advanced Manufacturing Technology*, vol. 49, no. 5-8, pp. 615–626, 2010.

[10] A. Milik, "On hardware synthesis and implementation of plc programs in fpgas," *Microprocessors and Microsystems*, 2016.

[11] S. Williams, "ICARUS Verilog at http://www.icarus.com/eda/verilog/," 2007.

[12] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin II - An Open-source Verilog HDL Synthesis tool for CAD Research," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 149–156. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/FCCM.2010.31

[13] J. Rose, J. Luu, C. Yu, O. Densmore, J. Goeders, A. Somerville, K. Kent, P. Jamieson, and J. Anderson, "The vtr project: architecture and cad for fpgas from verilog to routing," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, 2012, pp. 77–86. [Online]. Available: http://dl.acm.org/citation.cfm?id=2145708

[14] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to technology mapping for LUT-based FPGAs," *IEEE Transactions on CAD*, vol. 26, no. 2, pp. 240–253, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.9003

[15] GNU, "Bison - GNU parser generator," 2009. [Online]. Available: \url{http://www.gnu.org/software/bison/}

[16] Vern Paxson, "The Lex & Yacc Page," 2009. [Online]. Available: \url{http://dinosaur.compilertools.net/flex/index.html}

[17] Altera, *Quartus II Handbook, Volumes 1, 2, and 3*, 2004.