Hindawi
International Journal of Reconfigurable Computing
Volume 2022, Article ID 8827417, 10 pages
https://doi.org/10.1155/2022/8827417

Research Article

A Methodology for an FPGA Implementation of a Programmable Logic Controller to Control an Atomic Layer Deposition System

Peter Jamieson, Donald Blank, Janelle Ghanem, Tyler McGrew, and Giancarlo Corti

Miami University, Oxford, OH 45056, USA

Correspondence should be addressed to Peter Jamieson; jamiespa@miamioh.edu

Received 13 August 2020; Revised 25 October 2021; Accepted 30 March 2022; Published 6 May 2022

Academic Editor: Miriam Leeser

In this work, we present an industrial cold walled Atomic Layer Deposition (ALD) system, which can be controlled by either a traditional programmable logic controller (PLC) system or a field-programmable gate array (FPGA) prototyping board. This work presents an FPGA controlled system that takes ladder diagram (LD) control for a PLC and converts this control to Verilog HDL and programs an FPGA such that the FPGA prototyping board is used to control a real industrial application. We explore this approach since FPGA implementation of LD control could significantly reduce the cost of implementing these controllers with other potential advantages such as the improved granularity of timing control from milliseconds to nanoseconds, additional available pins for inputs and outputs far exceeding that of microprocessors, and lower power consumption for control. In this work, we provide details and descriptions of our industrial system (ALD), the LD control of this system and its implementation, our software flow to convert LDs to Verilog HDL, and our FPGA prototype board design to replace the existing electronic controller. We show how our LD-Verilog HDL converter in conjunction with FPGAs matches a PLC and demonstrate some of the benefits of using an FPGA.

1. Introduction

The five major manufacturers of Programmable Logic Controllers (PLCs) for controlling industrial manufacturing processes include Allen-Bradley, Siemens, Schneider Electric, Mitsubishi Electric, and Omron (https://www.businesswire.com/news/home/20160502005493/en/Technavio-Announces-Top-Vendors-Global-Micro-PLC accessed March 10, 2020). These companies tend to provide software for designing the controller, the hardware to execute the control logic with, and several tools for advanced features such as monitoring, safety, and data analytics.

One aspect of this space, which we believe can be improved, is the hardware implementation for the actual computational control. The invention and use of ladder logic (initially designed to control relays) to create these control systems are somewhat antiquated. The shift from relay-based control to programmable electronic implementation has resulted in the design of computational systems that enables the implementation of control algorithms for a wide range of applications.

PLCs consist of circuitry that senses inputs and generates control signals and a CPU that generates the appropriate signals for an industrial system to implement a control algorithm. The CPU implements this control using timers, counters, mathematical operators, internal state, and signals of both analog and digital inputs from the system. This logic control can be executed on a broad range of computational substrates other than a traditional CPU, and in particular for this work, an implementation of Field-Programmable Gate Arrays (FPGAs) has the potential to significantly reduce the cost of the controller and reduce the execution time of the process from the millisecond range to the nanosecond range. Not only is a small FPGA, costing less than 50 USD, more than capable of running the control logic specified in LD, but it is also designed to execute these types of designs efficiently. It can be wired into these systems, and it uses less power than a traditional microprocessor-PLC controller [1] (noting, however, that power is not a major issue in many industrial applications).

The goal of this paper is to show a real industrial system controlled via an FPGA. In particular, our contributions are as follows:

(1) A description and real implementation of the LD control into an industrial system, an Atomic Layer Deposition (ALD) system. This is the second, as we are aware of, demonstration of a real system being controlled via LD control that is converted and mapped to an FPGA.

(2) A tool (which is released open-source) to convert LDs into Verilog.

(3) A high-level method to map ladder logic control algorithms (using our open-source tool) to an FPGA prototyping board and the wiring and interfacing of that board into an industrial system. Though this may not be considered a major contribution, this workflow is key to achieving our second contribution and will help researchers and developers in the future.

(4) A demonstration of why FPGAs can improve the control of manufacturing as compared to traditional PLC controllers for the ALD system.

The remainder of this paper is organized as follows: Section 2 provides some background information and research, Section 3 describes the ALD system and the standard PLC implementation and wiring, Section 4 describes our software, the flow, and the wiring to use an FPGA to replace the PLC, Section 5 provides some results of our comparison of the two controllers, and Section 6 concludes the work and provides ideas for future work.

2. Background

This section reviews the basic concepts of an ALD system, ladder logic as ladder diagrams (LDs), and FPGA-based PLC implementations.

2.1. Atomic Layer Deposition System. The main idea of an Atomic Layer Deposition (ALD) system, first proposed by Aleskovskii [2], is to build up a physical material one atomic layer at a time by sequentially introducing gaseous precursors into the system that chemically modifies the material. Specifically, gases, called precursors, are introduced in a time-based sequence to create the material.

Puurunen [3] published a history of ALD systems, which is a nice introductory academic reference to this topic. More recently, Oviroh et al. [4] wrote a comprehensive paper on the state of ALD systems providing insights on new materials that this process may create and how advances in computation and experimentation are impacting this technology. In terms of the PLC integration with the ALD system, the control is not overly complex, and there has been little published research on this topic. Maula's research paper [5] briefly describes an industrial ALD system and the existence of the PLC controller. Additionally, there are several patents for this technology (for example, Patent 10,186,426), but we do not include those references here.

The details of our ALD system are provided in Section 3, including the LD algorithm.

2.2. Ladder Diagrams. The control of PLCs includes the use of ladder logic with what is called a ladder diagram (LD) and is formalized in the IEC61131-3 standard [6]. LD is a design language for describing control. Zhou and Twiss [7] provided an introduction explaining the workings of LDs compared to Petri nets. Also, for more information on LDs, there are many good basic introductions, for example, Hooper's book [8].

Figure 1 shows an example of an LD system that has three inputs called Start, *Stop*, and Input and three outputs called Out 0, Out 1, and Out 2. These signals are shown in the diagram, labeling either contacts (two parallel vertical lines) or coils (opening and closing round brackets). In this LD, there are three rungs of a ladder (Rung 0, Rung 1, and Rung 2 colored green, red, and blue, respectively), where the term ladder comes from how the LD looks like a ladder.

Processing LD is done from top to bottom; in Figure 1, Rung 0, Rung 1, and Rung 2 are processed in that sequential order, and the processing of all the rungs is known as a "scan." For Rung 0, we can interpret it as a logical AND, where if both Input and Start have been pressed/activated, then the coil Out 0 generates an on signal. Once Rung 0 is evaluated, we next evaluate Rung 1, but the inputs for Rung 1 are the outputs of both **Rung 0** and Rung 2 organized as a logical OR operation. Since Rung 2 has not been evaluated yet, we use the value of Out 2 from the previous scan. From a finite state machine's perspective, we would say we use the previous state value of Out 2. It is important to understand that LD proceeds from top to bottom as it suggests that an LD needs to be sequentially evaluated to maintain the designer's intended functionality, but if there are no dependencies in a rung, then, in theory (and as described by Milik [9]), the rung can be calculated in parallel with other rungs. Finally, the last rung is a latch (implemented with an OR followed by an AND); in that, once the Start input is activated, the Out 2 signal will stay on until a Stop signal is pressed.

Other special-purpose functions (called block functions or boxes by Siemens) can be defined in LDs, such as timers, counters, conditionals, and more complex finite state machines. Our work uses some of these structures, in particular timers and counters, and we will define them as they appear in our designs.

2.3. FPGAs as PLCs. FPGAs are a type of Integrated Chip (IC) that is reconfigurable devices, meaning that the functionality can be programmed or, possibly, reprogrammed after being manufactured. This means that these ICs can be programmed to implement a digital design. The FPGAs programmability, compared to ASICs, comes at a cost in terms of speed, power, and area [10], but to create an IC for a specific application is only viable for mass markets due to the high cost of the nonrecurring engineering cost in the millions to hundreds of millions of dollars. FPGAs are a common approach to designing digital systems in low-volume markets and for prototyping designs. For these reasons, an FPGA might be an appropriate device that could replace the PLC, and FPGAs only cost from tens of dollars to

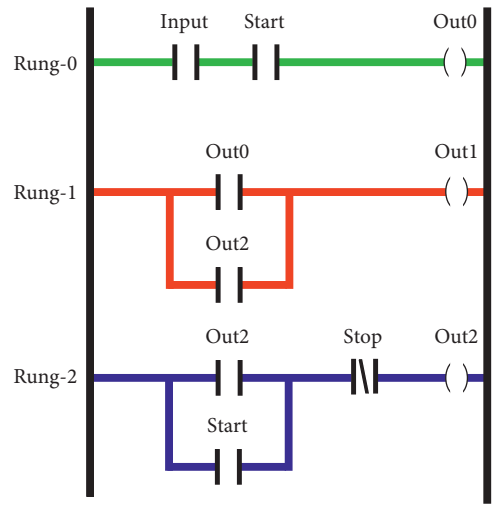

Figure 1: A simple LD example in diagram form.

tens of thousands of dollars depending on capacity and speed requirements. Intel and Xilinx are the two largest FPGA companies.

Researchers have and are still working on how reconfigurable devices can be used to implement LDs. The first implementation of an alternative control for Programmable Logic Devices (PLDs) was by Adamski and Monteiro [11]. The goal was to implement control on a PLD and to convert existing PLC designs into a PLD. They followed up this work with a later publication [12] that improved on their original work. Also, this work was further extended with Wegrzyn et al. [13], where they used the Petri net descriptions, but in this case, mapped their design to VHDL [14] so that they could target commercial CAD flows which map to devices such as FPGAs and Application-Specific Integrated Chips (ASICs).

Petko and Karpiel [15] 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 an autonomous flow, but they provide many steps, both automated and manual, to map their control designs to a silicon implementation in either Verilog or VHDL. The input to their system, however, is not LD, and instead, the complete control is designed by an engineer familiar with their flow.

Economakos and Economakos [16, 17] 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 which is then converted by the Catapult C tool flow (created by Mentor Graphics) to target an FPGA. Their main motivation is to efficiently implement more complex control designs that may not be feasibly implemented on a PLC.

Du et al. [18] created a tool to map LDs to VHDL. Their motivation is to improve performance by using an FPGA, and they target Xilinx FPGAs. The limitation with their work is the tool, which is not open-source, and there has been no continuance or commercialization of their original work. Milik [9] created a flow from IEC61131-3 LDs to FPGAs via Verilog as the CAD flow design entry language. Milik spends significant time on optimizations in the conversion using compile-time approaches such as building a directed flow graph and analyzing these graphs. More recently, Milik et al. have furthered our understanding of compiling LD to FPGAs by direct FPGA synthesis [19].

Ichikawa et al. [20] also demonstrate their conversion tool from LDs to VHDL and map their hardware design to Altera FPGAs. This work demonstrated a real FPGA interfaced with a real "perfect layer winder," and they are the first group to demonstrate a full and real system controlled via an FPGA-based controller. Additionally, they created a custom FPGA prototyping board for this work since they could not find a board on the market that satisfied their needs.

In Corti et al. [21], we created a tool to take Allen-Bradley LD and convert it to synthesizable Verilog targetting an FPGA. Similarly, Eassa et al. [22] implemented a RISC-V processor on an FPGA that can execute the control logic.

3. Implementation of the ALD System

A schematic of the general ALD setup is depicted in Figure 2. The ALD reactor used for all experiments is a cold wall thermal ALD reactor, meaning that all energy required for the reaction is provided by heat, as opposed to plasma/radical-enhanced ALD [23]. The reactor uses an inert carrier gas (N_2) to carry the precursors into and out of the chamber. Although the use of carrier gases results in less efficient utilization of precursors, it also allows for shorter purge times to effectively clear the chamber, allowing for shorter cycle times overall [23].

3.1. Control of Our ALD System with PLC. The ALD sequence begins by pressing the start button (start), which prompts the vacuum valves VV1 and VV2 to open and start the controllers of the precursor bottles and reactor heater (Pbot and RH), respectively. When the chamber pressure (Pstch) is below 1 mbar, the controller turns on the mass flow controllers (MFC) to continuously flow N_2 until the end of the deposition sequence. With the N_2 flowing, the ALD waits until the PIDs signal the controller that the bottle (TP1) and reactor (TR) temperatures reach their setpoint. When the system is up to temperature, the deposition cycle shown in Figure 3 begins by closing the VV1 and VV2 valves. Then, the ALD controller follows the deposition pattern; first, it activates precursor valve (SV1) for a precursor pulse of tpp1 msec, waits for twp1 msec, and activates vacuum valve one (VV1) for tvp1 msec; when the VV1 closes, it activates the water valve (SV4) for a water pulse of tw msec, waits for tww msec, and then activates vacuum valve two (VV2) for tvw msec, and when the VV2 closes, the controller counts one cycle. Finally, when either the desired number of cycles have occurred or the stop button (stop) is pressed, the controller will shut down the heaters Pbot and RH and close the vacuum valves VV1 and VV2. The N_2 must flow until the pressure in the chamber equals to atmospheric pressure

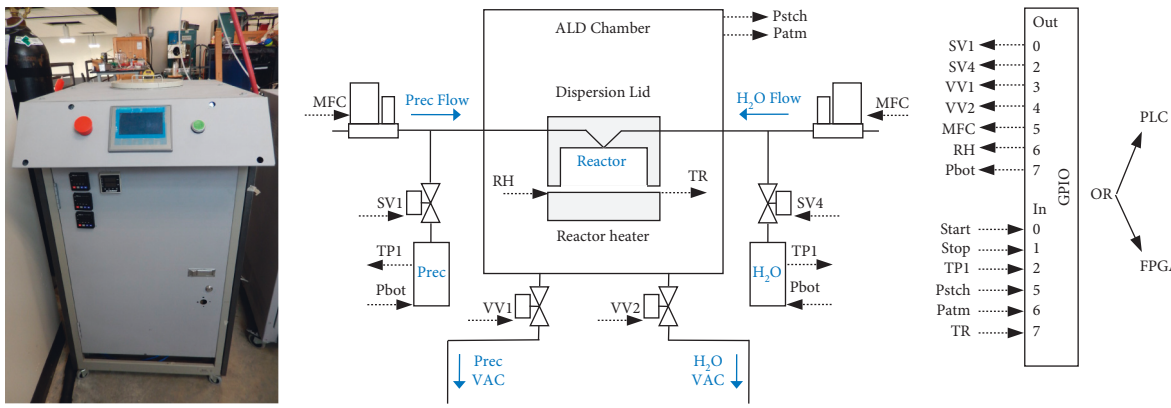

FIGURE 2: An image of our ALD system (a), schematic with valves and sensors (b), and corresponding signals as a general-purpose I/O pin layout (c).

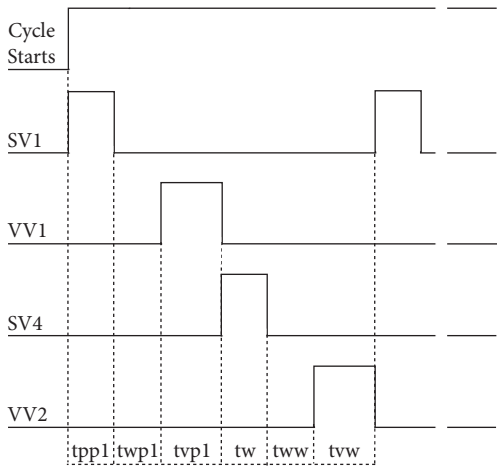

FIGURE 3: A timing diagram of the valve signals for our ALD system.

(Patm). We have implemented this algorithm as an LD design with 37 rungs for both Allen-Bradley PLCs and Siemens PLCs, where the first is available to view on our GitHub repository (https://github.com/ghanemja/senior-design).

Figure 2 shows a picture of the ALD system at Miami University. The picture gives researchers a view of what this type of manufacturing system is and how it is interfaced. The main chamber is held within the cabinet. Additionally, the front panel has a red emergency stop button, a human-machine interface (HMI), and another button to start operation. When the FPGA is wired into the system, the PLC (also contained in the cabinet) is disconnected, and the FPGA interface replaces this controlling connection.

4. Replacing a PLC with an FPGA

The two primary steps required to convert an LD to work on an FPGA to control an industrial system are as follows: first, converting the LD into a form that can be inputted and passed through an FPGA CAD flow to program the FPGA, and second, a wiring interface between the FPGA and the industrial system to properly send the electrical signals to the system.

4.1. Converting LDs to Verilog HDL. In previous work, we created Hashigo and released the open-source code at https://github.com/NigoroJr/hashigo that was a complete LD to Verilog conversion system using Flex and Bison to parse the LD and back-end code that compiles the LD to Verilog [21]. We, however, learned that a complex full compiler approach is not needed to process LDs to Verilog. Instead, in this version of our work, we present a complex Python script that can perform the same conversion process and has support for timers and counters, the latter of which Hashigo could only implement via alternative methods.

Algorithm 1 shows the high-level steps the Python script and the associated environment take to convert LDs to Verilog files. Note that specific details, such as read, write, and processing details of the conversion, can be seen in the open-source code: https://github.com/ghanemja/senior-design. This file type is structured in XML format, shown in Figure 4, and contains all the information about an Allen-Bradley ladder logic program and environment needed to convert the program to HDL. This .L5X file is parsed using the ElementTree XML python library to obtain the root of the XML tree.

Using an XML input format, shown in Figure 4, from the LD programming environment and Algorithm 1, we parse each rung of the ladder sequentially and convert the rungs into a simplified, intermediate data structure, as shown in Figure 5. Once this intermediate file exists, it is then converted into a Verilog file that represents the LD, shown in Figure 6. This logical Verilog expression is created by using a dictionary or look-up table, shown in Table 1, where each node type corresponds directly to a string segment which is added to the overall expression string, as shown in Figure 6.

This approach interprets each LD rung sequentially and creates the Verilog assignment statements. Then, the system creates an extra rung to assign each register's memory value

```
(1) function Ladder2Verilog(L5X_filename)
(2) L5X_File = read(L5X_file.l5x)
(3) Find physical IOs, logical rungs, and function modules information
(4) write(Hashigo_File.hshg)
(5) Hashigo_File = read(Hashigo_File.hshg)
(6) Verilog_File = read(Verilog_Template.v)
(7) Assign physical IOS to FPGA pins
(8) for all rungs do
(9)     Convert rung to nodes_list
(10)    Make output_functions list from nodes_list
(11)    Create Verilog logical_expression from nodes_list
(12)    Construct Verilog rung_text assignment statements
(13) end for
(14) Add rung assigning memory registers to outputs
(15) Create register declarations and reset assignments
(16) Create function module instantiations
(17) for all placeholders do
(18)    Verilog_File = Verilog_File.replace(placeholder, text)
(19) end for
(20) write(Verilog_File.v)
(21) end function
```

Algorithm 1: Pseudocode for steps taken to convert LDs to Verilog HDL.

```
<Text>
<![CDATA[[XIO(Local:2:I.Data.1) ,XIC(cycles.DN) ]OTE(mstop);]]>
</Text>
```

Figure 4: An example of a logical rung found in an L5X XML tree.

```
START []
LBRACKET []
XIO [Stop]
OR []
XIO [Start]
AND []
XIC [cycles_DN]
RBRACKET []
AND []
OTE [mstop]
END []
```

Figure 5: An example of the logical rung converted from XML to the intermediate data structure.

```
0: begin
    n_mstop <= (!n_Stop || !n_Start && n_cycles_DN );
end
```

Figure 6: Verilog assignment statement created from the intermediate data structure.

and the register declarations and assignments based on the information from the date structures previously created. Then, the system creates module instantiations for each function module used in the ladder logic (timers, counters, etc.) and wires these modules into the main module. Finally, the system uses all of the text sections to write the final Verilog file to the directory. Our Python approach is easy for researchers to understand and extend, and our controller implementation on an FPGA results in an implementation that is fast enough to process the control required by the industrial applications we are targeting.

Table 1: Dictionary of ladder logic functions with their Verilog equivalents.

LD function	LD XML	Verilog text
and	and	&&
or	or	
NO contact	xic(a)	a
NC contact	xio(a)	!a
Comparator ≥	GEQ(a, b)	(a≥b)
Comparator ==	EQu(a, b)	(a==b)
One shots relay	ONS(a)	(a&&!prev_a)
FB timer	TON()	Timer()
FB counter	CTU()	Counter()

4.1.1. Timer and Counterimplementation Details. Our previous research and software in this space, Hashigo, supported LD conversion, but it did not include the implementation of timers and counters. However, our tool flow in the Python version has a way of implementing both timers and counters, and we added this functionality since we needed both for interfacing with the ALD system.

To implement timers or counters (or, in theory, any functional block), we implement a configurable Verilog module that has inputs and outputs of a timer, where the inputs can be set to parametrize the needed functionality. The Python scripts read the type of functional block in the LD and set the appropriate signals. These blocks have been previously added to the dictionary or look-up table, shown in Table 1. Specific details on the generation of these function blocks can be found on the open-source code: https://github.com/ghanemja/senior-design. Our system, however, targets a Verilog implementation but is not provided with the

clocking speed of the FPGA prototype board, and therefore, the designer of the FPGA control system will have to do some calculations to set up the parameters for the required timer rate.

4.2. FPGA Prototype Board as a Controller. Once the Verilog files have been generated by the script, the next step is to map the design onto an FPGA and interface the FPGA with the system replacing the PLC. Our replacement PLC is a DE2-115 FPGA prototyping board from Terasic that includes a Cyclone IV Intel FPGA. This board is commercially available and used by universities and undergraduate students to learn how to use FPGAs.

To map the Verilog files onto the FPGA, we use a commercial CAD tool that converts design files into programmable bit files that, when loaded on a specific FPGA, implements the logic design on the FPGA. Since we are using Intel FPGAs, the corresponding Intel tool is Quartus [24]. Within Quartus, we create a project, set up the appropriate Verilog design files, and most importantly, map the pins of the FPGA to particular signals in the design. Pin-mapping maps the inputs and outputs of the ALD system to the pins of the FPGA controller, and on the DE2-115 board, the general-purpose input-output (GPIO) pins allow for these connections.

Once the digital signals are mapped to the GPIO pins, the next step is to take the three to five volt signal with low amperage output and convert the control signals to the industrial standard 24 volts [25]. Once the signals are conditioned to meet the ALD input and outputs, we can connect the FPGA board into the system to control it.

Figure 7 shows the printed circuit board (PCB) developed to interface the FPGA GPIOs with the ALD systems' circuitry. Since both ALD and the FPGA use a 40-pin connection, we developed this PCB to interface the two systems with two integrated circuits (IC) components.

To convert the 24 V digital of the ALD signal to 3.3 V level for FPGA input, a TA ULN2003AN IC is used. This 7-channel Darlington transistor array, commonly used for switching applications, is operated in an inverting, common-emitter configuration, as can be seen in Figure 8(a), where V_{IN} is a 0–24 V digital signal from the ALD, V_{CC} is 3.3 V, and R_L is a digital input to the FPGA.

However, this low-side driver configuration cannot be used to drive ALD inputs (solenoid valves, relays, etc.) from FPGA signals because this topology is not able to deliver the required 400 mA of current to inductive loads, Z_L. Instead, a high-side driver topology based on the MIC2981 IC is used to drive the ALD inputs from the FPGA, as shown in Figure 8(b), where V_{IN} is a 0–3.3 V digital signal from the FPGA, V_{CC} is 24 V, and Z_L is an ALD valve coil.

5. Results and Testing

For our experiments, we run two tests. First, we show that the FPGA implementation is equivalent to the PLC by comparing the signals generated by both systems and verifying them. For our second result, we show how the

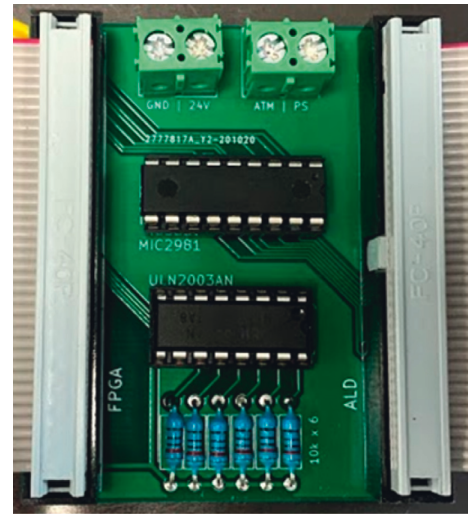

Figure 7: Our PCB developed for interfacing the FPGA with the ALD.

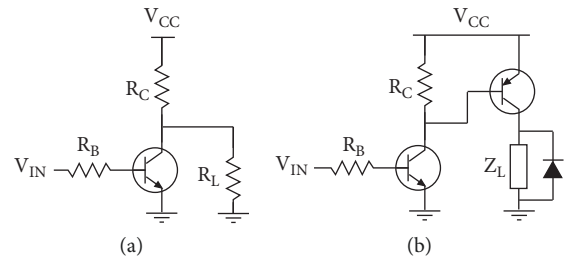

Figure 8: Simplified switching topologies used for interfacing the FPGA and ALD: (a) the switch used as an inverting, low-side switch to convert 24 V ALD signals to 3.3 V FPGA signals and (b) the switch used as a noninverting high-side switch to drive 24 V inductive ALD loads from a 3.3 V FPGA signal.

FPGA has the potential for finer timing-granularity compared to a PLC. Finally, we describe our complete system where the FPGA controls the ALD system in a real-world test.

5.1. ALD Equivalent Control with FPGA Prototyping Board. In our first experiment, we show how the signals generated by the FPGA (which are mapped by our Python scripts and used to create the design files that are programmed to the FPGA) are timing equivalent to the PLC-generated signals programmed from the Allen-Bradley tool and mapping flow.

Figure 9 shows a screenshot of our comparison method. The full video can be viewed at https://www.youtube.com/watch?v=cyB6Dcb55wc, and the comparison of the FPGA (top) and PLC (bottom) starts at timestamp 4 : 30 of the video. In Figure 9, we can see that the FPGA has lit two of the LEDs on the FPGA prototyping board, which matches the two LEDs, 6 and 7, on the PLC. In the film, we show how the prototyping board LEDs match the PLC lights. This is a functional demonstration of the equivalency of the two controllers, and it also shows how our alternative flow that maps to the FPGA is equivalent to the PLC program noting

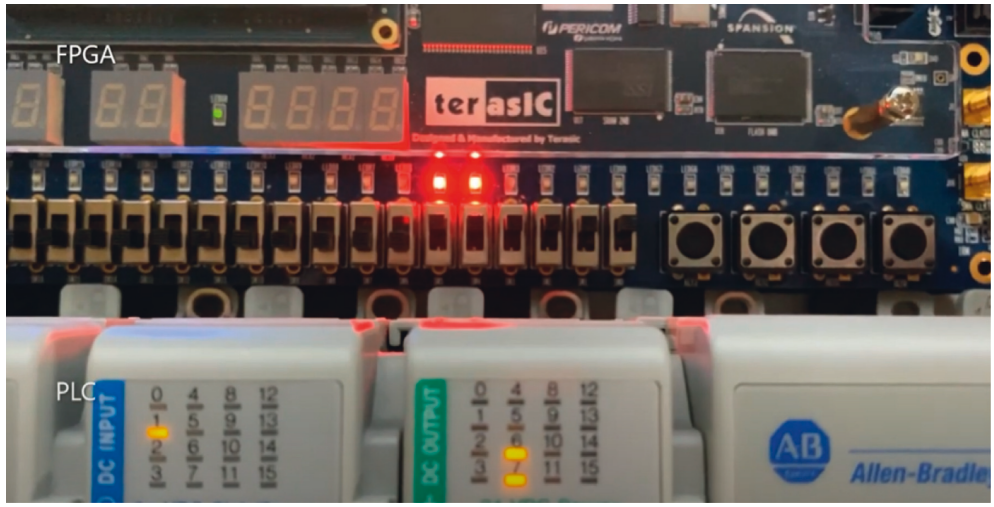

Figure 9: A comparison of the output signals on an FPGA (top with red LEDs) to the PLC controller (bottom with orange LEDs).

that we use the same LD, our Python tool converts this LD design to be implemented on the FPGA on the DE2-115.

The current processing cost on the FPGA (which we call a duty-cycle, equivalent to a PLC scan) is 37 rungs plus one cycle to update all the outputs. On the FPGA, we execute this; this means that the duty-cycle is 760 ns. Note that this duty-cycle could be decreased if we included the optimization capabilities to analyze the dependencies between rungs and parallelizing independent rungs. However, the current scan of 760 ns is already fast compared to the PLC emulation.

5.2. Finer Grain Timing Implementation on an FPGA. Next, we show how the granularity of control timing possible on the FPGA is significantly faster than what PLCs are currently capable of doing. Most PLCs smallest granularity of timing is one millisecond to implement a toggling signal (clock). On the DE2-115, it is possible to implement a granularity of toggling that is equal to the clock speed of the FPGA.

The clock on the DE2-115 prototyping board is 50 MHz, meaning the clock period is 20 ns, and the granularity of the timing toggle would be 20 ns or 10,000 times faster than the 1 ms granularity of the PLC (note that the phase-locked loop functional units on this FPGA allow us to control the clock, but this is unnecessary). We note, however, that not many mechanical and manufacturing systems need this small granularity of control, but it is possible with this capability that new applications for this could be introduced within the field.

5.3. Real FPGA as Controller for ALD. The testing setup for the full ALD system with the FPGA controller is shown in Figure 10. An interfacing PCB (previously described) is connected to the FPGA and the ALD using two 40-pin ribbon cables, which fit into the 40-pin connectors on the PCB. Two wires are used to connect 24 V and ground from the ALD to the PCB through screw terminals. The 3.3 V

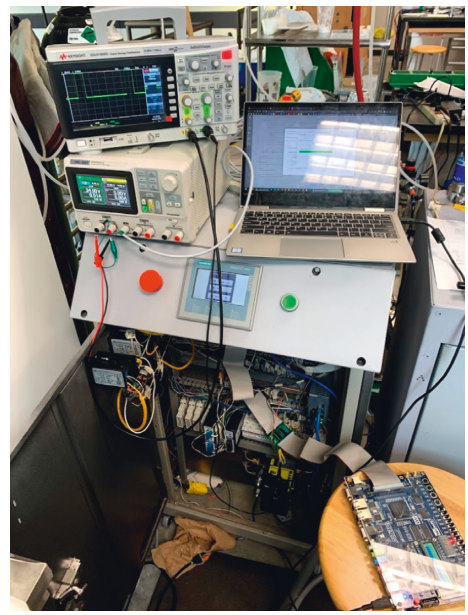

Figure 10: Our ALD system interfaced with the DE2-115 (bottom right).

and ground from the FPGA are connected to the PCB through two of the connections on the 40-pin ribbon cable. A BI1-EG05-AP6X inductive proximity sensor is attached to the SV1 valve to measure its physical actuation. This valve was powered at 24 V using a DC power supply. Finally, a Keysight oscilloscope is attached to both the FPGA control signal and the valve sensor to compare the intended and physical response of the SV1 valve within the ALD system.

With the hardware integration between the FPGA and ALD completed, we test the system to determine the capabilities of the FPGA. Initially, valve timings with the ALD cycle are programmed in seconds. This works well when simulating the outputs of the FPGA in simulation tests; however, since we could not physically see the outputs of the PLC and FPGA turning on and off at speed, we were unsure

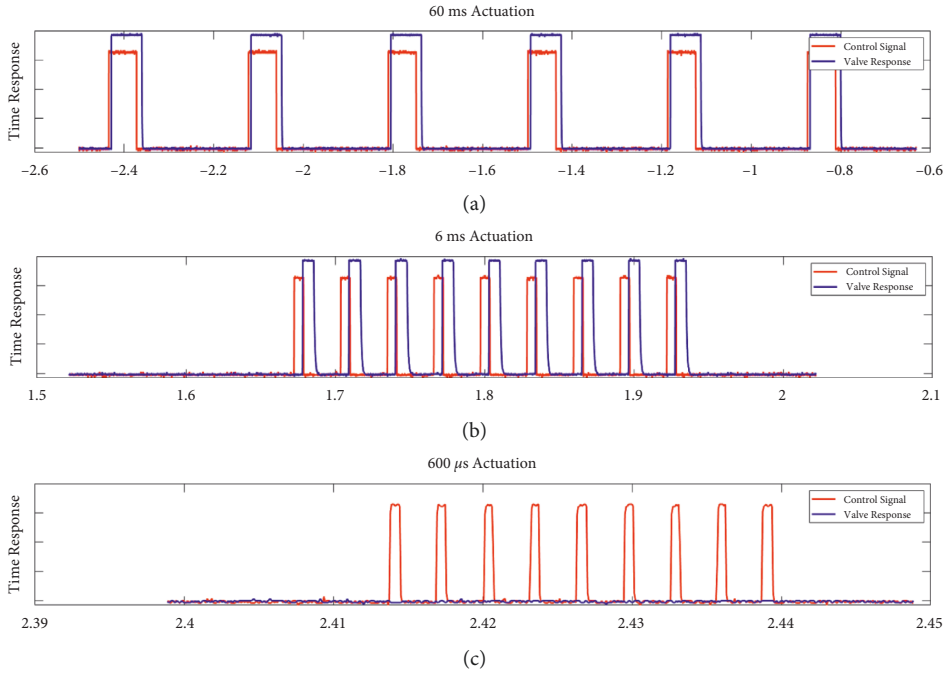

FIGURE 11: Control and response of the valve for decreasing actuation times: valve actuated for (a) 60 ms, (b) 6 ms, and (c) 0.6 ms.

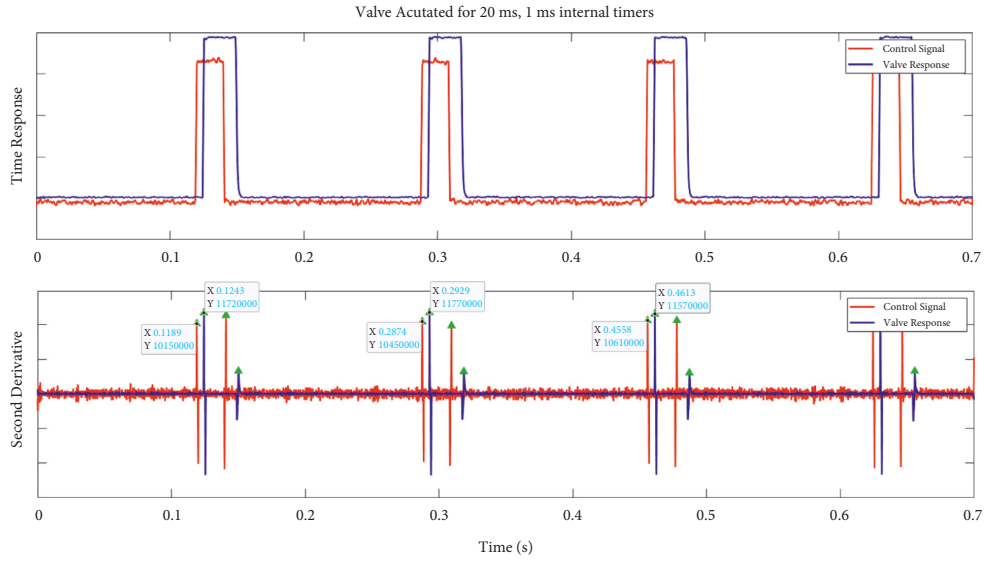

FIGURE 12: Control and response of the valve when using 1 ms internal timers. Rising and falling edge times are labeled in the bottom of the two timing waveforms.

how the system would respond once switching speed is increased. To test whether the system is operating as desired, we attached an oscilloscope probe to the output of SV1 on the FPGA and a valve position sensor on the corresponding valve on the ALD. This valve is the first valve that is opened during the ALD cycle (as can be seen in Figure 2), so measuring its response allows us to determine the cycle time of the ALD by measuring the rising edge to rising edge period of the valve position feedback. The valve position sensor is an on/off switch, and the rising edges are easy to identify in the waveforms.

We measure the response time of the ALD valve to determine what is the minimum cycle time the valve can be operated at. Figure 11 shows the switching waveform in red (lower peaks) and the valve's physical response to a switch in blue (larger peaks). As can be seen in Figure 11, once valve timing is at 5-6 ms (Figure 11(b)), the valve can still respond, but at a switching speed of 600 μs, the valve fails to respond (Figure 11(c)). Therefore, to ensure that the system is stable, valve timing on the FPGA is set to 20 ms.

Next, to show what the FPGA controller is capable of compared to a PLC, we reduce the timing of the internal

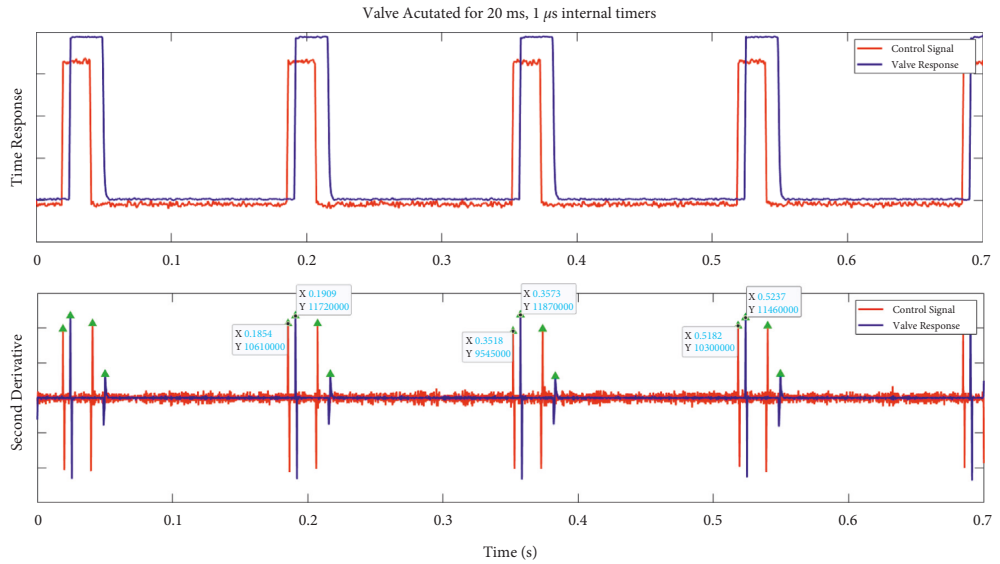

FIGURE 13: Control and response of the valve when using 1 μs internal timers. Rising and falling edge times are labeled in the bottom of the two timing waveforms.

timers within the ALD cycle called "twait1" and "twaitw" first to the minimum time that the PLC can control them, which is 1 ms. The cycle time for the ALD cycle is recorded as the peak-to-peak timing of the rising edge of the valve response, as can be seen in Figure 12. The ALD cycle time (a complete control cycle) takes 168 ms when controlled by the PLC.

By changing these two internal timers so that they are faster than what the PLC is capable of, we record the ALD cycle time for the FPGA. As shown in Figure 13, the internal timers are reduced to 1 μs, which results in a 2 ms reduction in one ALD deposition cycle. For example, a 50 nm copper oxide layer deposited via ALD has an average growth per cycle of ≈ 0.15 [26], which represents a 3.3 sec reduction per batch. This shows that an FPGA controller (even a simple, cheap one) is able to improve the overall production time a PLC controller can achieve. Note, however, that we are showing these results to demonstrate the capability of an FPGA-based controller and not stating that this would be recommended for the ALD system.

6. Conclusion

This work demonstrates how an ALD system is controlled with an FPGA prototyping board that replaces the typical PLC. To achieve this, we describe our ALD system, its control requirements, and the tools we have created to automatically convert LD (the design method for PLCs) into Verilog files that can then be synthesized and programmed to an FPGA. The FPGA with electronic conditioning of signals can then be used to replace the existing PLC, and we show that we can match the control signals designed in LD for a PLC to the FPGA. This means that our ALD system can be controlled via this system.

Additionally, we show that the potential for using an FPGA instead of PLC systems has the capability to increase the granularity of control. Note that the parallel capabilities and number of pins available on the FPGA mean that these devices have great potential to revolutionize control in manufacturing. This benefit comes at no cost as the traditional design flow used for these devices can be used, and our flow can convert the designs to the FPGA. Additionally, FPGAs are off-the-shelf components that are not overly expensive.

The future of the FPGA as a standard device in the control systems is promising; however, FPGAs are difficult to design for, and manufacturing engineers are not trained in their use. We have shown, in two cases, that if LD continues to be the design entry system of choice, then it is not too difficult to build tools to convert these designs to map to an FPGA. The real future of this work is for either a PLC company or a startup to embrace the idea of the FPGA, create a tool flow, and generate prototyping boards and breakout boards with a focus on PLC controlled manufacturing.

Data Availability

There are no significant data in this work, but all design files are available at https://github.com/ghanemja/senior-design.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] P. Jamieson, T. Becker, P. Y. K. Cheung, W. Luk, T. Rissa, and T. Pitkänen, "Benchmarking and evaluating reconfigurable architectures targeting the mobile domain," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, no. 2, pp. 1–24, 2010.

[2] V. Aleskovskii, *Matrix hypothesis and way of synthesis of some active solid compounds*, PhD Thesis, 1952.

[3] R. L. Puurunen, "A short history of atomic layer deposition: tuomo suntola's atomic layer epitaxy," *Chemical Vapor Deposition*, vol. 20, no. 10–12, pp. 332–344, Wiley, Hoboken, NJ, USA, 2014.

[4] P. O. Oviroh, R. Akbarzadeh, D. Pan, R. A. M. Coetzee, and T.-C. Jen, "New development of atomic layer deposition: processes, methods and applications," *Science and Technology of Advanced Materials*, vol. 20, no. 1, pp. 465–496, 2019.

[5] J. M. Jarmo Maula, "Atomic layer deposition for industrial optical coatings," *Chinese Optics Letters*, vol. 8, no. S1, pp. 53–58, 2010.

[6] M. Tiegelkamp and K. H. John, *IEC 61131-3: Programming Industrial Automation Systems*, Springer, Berlin, Germany, 2010.

[7] M. C. Zhou and E. Twiss, "Design of industrial automated systems via relay ladder logic programming and petri nets," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 28, no. 1, pp. 137–150, 1998.

[8] J. Hooper, *Introduction to PLCs*, Carolina Academic Press, Durham, NC, USA, Second Edition, 2006.

[9] A. Milik, "On hardware synthesis and implementation of PLC programs in FPGAS," *Microprocessors and Microsystems*, vol. 44, pp. 2–16, 2016.

[10] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 203–215, 2006.

[11] M. A. Adamski and J. L. Monteiro, "PLD implementation of logic controllers,"vol. 2, pp. 706–711, in *Proceedings of the IEEE International Symposium on Industrial Electronics*, vol. 2, pp. 706–711, IEEE, Athens, Greece, July 1995.

[12] M. Adamski and J. L. Monteiro, "From interpreted petri net specification to reprogrammable logic controller design,"-vol. 1, pp. 13–19, in *Proceedings of the 2000 IEEE International Symposium on Industrial Electronics (Cat. No.00TH8543)*, vol. 1, pp. 13–19, IEEE, Cholula, Mexico, December 2000.

[13] 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.

[14] IEEE, *IEEE Standard VHDL Language Reference Manual*, IEEE, Manhattan, NY, USA, 1987.

[15] M. Petko and G. Karpiel, "Semi-automatic implementation of control algorithms in ASIC/FPGA," in *Proceedings of the 2003 IEEE Conference on Emerging Technologies and Factory Automation*, pp. 427–433, Lisbon, Portugal, September 2003.

[16] C. Economakos and G. Economakos, "FPGA implementation of PLC programs using automated high-level synthesis tools," in *Proceedings of the IEEE International Symposium on Industrial Electronics*, pp. 1908–1913, IEEE, Cambridge, UK, June 2008.

[17] C. Economakos and G. Economakos, "Optimized FPGA implementations of demanding PLC programs based on hardware high-level synthesis," in *Proceedings of the 2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1002–1009, IEEE, Hamburg, Germany, September 2008.

[18] 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. 5–8, pp. 615–626, 2010.

[19] A. Milik, M. Kubica, and D. Kania, "Reconfigurable logic controller-direct FPGA synthesis approach," *Applied Sciences*, vol. 11, no. 18, p. 8515, 2021.

[20] S. Ichikawa, M. Akinaka, H. Hata, R. Ikeda, and H. Yamamoto, "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.

[21] G. Corti, D. Brunner, N. Mizuno, and P. Jamieson, "Transforming ladder logic to verilog for fpga realization of programmable logic controllers," in *Proceedings of the International Conference on Embedded Systems, Cyber-Physical Systems, and Applications (ESCS)*, pp. 35–38, Las Vegas, NV, USA, July 2017.

[22] H. Eassa, I. Adly, and H. H. Issa, "RISC-V based implementation of programmable logic controller on FPGA for industry 4.0," in *Proceedings of the 2019 31st International Conference on Microelectronics (ICM)*, pp. 98–102, IEEE, Cairo, Egypt, December 2019.

[23] S. M. George, "Atomic layer deposition: an overview," *Chemical Reviews*, vol. 110, no. 1, pp. 111–131, 2010.

[24] Altera, *Quartus II Handbook: Volumes 1, 2, and 3*, Intel, Santa Clara, CA, USA, 2004.

[25] J. C. Thompson and D. B. Durocher, "24 V DC control-an emerging alternative to legacy 120 vac control applications in north America," in *Proceedings of the Conference Record of the 2002 Annual Pulp and Paper Industry Technical Conference (Cat. No.02CH37352)*, pp. 70–75, Toronto, Canada, June 2002.

[26] T. Iivonen, M. J. Heikkilä, G. Popov et al., "Atomic layer deposition of photoconductive Cu_2O thin films," *ACS Omega*, vol. 4, no. 6, pp. 11205–11214, 2019.