Software Design of Veriloghdl Code Generation for Ladder Diagram and Data Acquisition Using LABVIEW

G Dhanabalan (dhanabalang@yahoo.com)
AAA College of Engineering and Technology https://orcid.org/0000-0003-3462-7920
Tamil Selvi S
National Engineering College Department of Electronics and Communication Engineering

Research Article

Keywords: ladder diagram, PLC, FPGA, scan time, verilog

Posted Date: July 13th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-663333/v1

Software design of VerilogHDL code generation for ladder diagram and data acquisition using LABVIEW

G Dhanabalan, Department of Electronics and Communication Engineering, AAA College of Engineering and Technology, Sivakasi, Tamilnadu, India, dhanabalang@yahoo.com

S Tamil Selvi, Department of Electronics and Communication Engineering, National Engineering College, K.R.Nagar, Kovilpatti, Tamilnadu, India, tamilgopal2004@yahoo.co.in

Abstract

Powerful advantages of programmable logic controller (PLC) dominate process industries. Scan time of PLC increases with the number of inputs, rungs added in ladder diagram (LD). Researchers have identified and proved that field programmable gate array (FPGA) is more suitable than PLC for high speed applications. PLC executes the instructions represented through LD. PLC programmers are not familiar with FPGA programming. But, FPGA does not support LD based programming. This work has developed application software to generate equivalent VerilogHDL code for LD using LabVIEW. Novelty in this work is that each rung is defined using an "assign" statement which helps simultaneous execution of all the rungs. A data acquisition system was created to monitor the digital signals handled by the FPGA. The software was verified with a case study of substances mixing and traffic light control system.

Key words – ladder diagram; PLC; FPGA; scan time; verilog;

1. Introduction

Programmable logic controller (PLC) is best utilized for industrial automation by many process industries [1 – 2]. Industrial automation views the process industry in terms of number of analog and digital signals to be processed. Fig. 1 shows the establishment of digital input module (DIM) and digital output module (DOM) with PLC. Digital input module uses multiplexing technique to receive digital inputs from the plant and store them in memory. A suitable data transfer technique transfers these digital data to PLC. Expansion connector connects more number of DIMs to PLC [3 – 4]. In one way, this option shows the capability of PLC in processing higher number of digital signals. But, it actually increases scan time which indirectly impacts the process condition. PLC uses these data whenever it executes ladder diagram (LD). The time required for this entire process is not convincing for high speed applications since the program executed by PLC is sequential [5 – 6]. The LD faces problems like data dependency [7], output dependency, missing of input signals and increase in logic scan time along with the increase in the number of rungs.

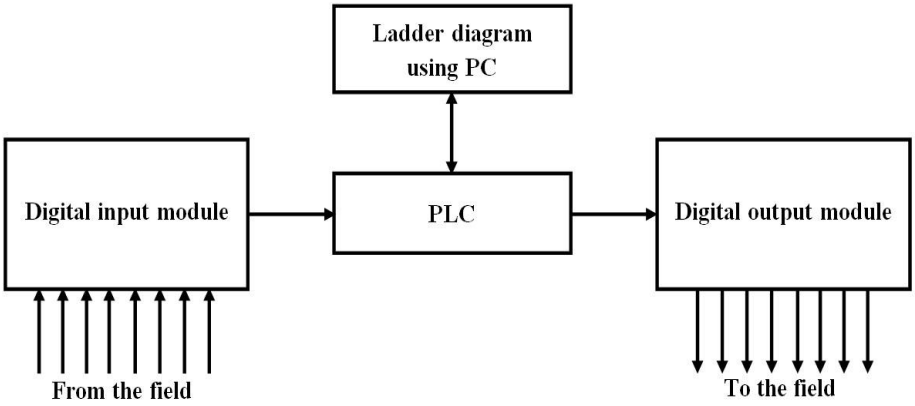


Fig. 1 PLC connection with digital inputs and outputs

LD in Fig. 2(a) is an example for data dependency problem. Output2 depends on the current value of Output1 which will be available only after the execution of first rung. Assume that the PLC generates Output1 and Output2 at every 25^{th} and 30^{th} milliseconds respectively. Result of Output2 will be wrong if the Input1 changes between 25^{th} and 30^{th} milliseconds.

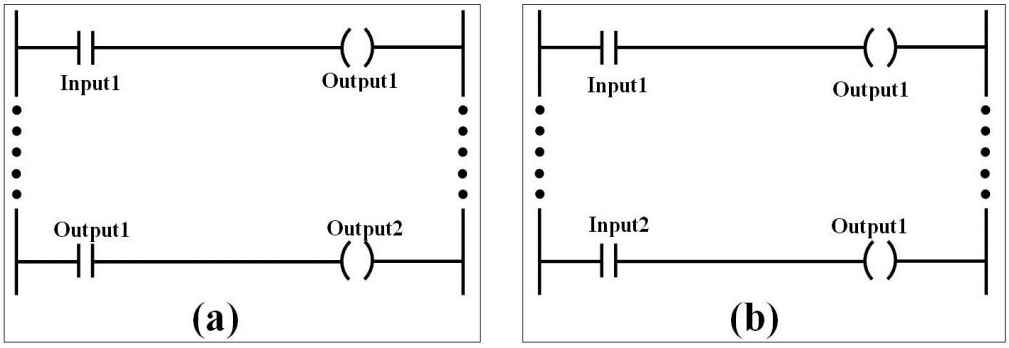


Fig. 2 Example ladder diagram for data dependency and output dependency

In case of output dependency, there could be a situation in which an output depends on the activation of any one of the input as shown in Fig. 2(b). When the Input1 is at high state, the Output1 is also in high state. However, the status of Output1 is based on the condition of Input2 also. In fact, it is the programmer's mistake wherein both Input1 and Input2 shall be connected in parallel and the output to Output1. Still, output dependency problem occurs. All such problems lead to poor processing condition.

The problems of data dependency, output dependency can be eliminated by using field programmable gate array (FPGA) to execute LD in terms of Hardware description Language (HDL) code as the execution is parallel and hardware based [8]. The effect of increase in number of digital inputs, number of rungs has zero impact in the execution time. Table 1 compares the advantages of using FPGA with processor in a process.

Table 1: Comparison of PLC used for ladder diagram execution & FPGA used for digital logic execution

Process using processor	Process using FPGA
Processor fetches digital signals from memory (Alive). It fetches only one signal at a time.	FPGA connects digital signals directly to the inputs / outputs (Live). All the signals are ready for concurrent execution [8].
Time required to execute a rung is the time required to store data in memory from fields, fetch all the necessary data from memory, execute the logic and store the output data in memory.	Delay time of the logic is the time required to execute a rung.
It executes rungs in sequential manner.	It converts all the rungs into digital logic and hence establishes simultaneous execution.
It stores ladder logic execution output(s) in memory one by one.	It stores ladder logic execution output(s) in Block RAM (Random Access Memory) simultaneously.
From memory, it forwards the data to digital output module(s) sequentially.	There is no need for digital output module(s) since FPGA has the capacity to directly connect all the outputs to the field devices.
Processor demands DIM(s) and digital output module(s) to process digital signals.	FPGA eliminates the need of DIM(s) and digital output module(s).

Even though FPGA is a better alternative for PLC, industrialists waver to replace it with latest technology [9 – 10]. PLC programmers are familiar with ladder logic program only. FPGA cannot process LD.

Researchers have already focused this problem and developed embedded system based solutions [11]. Since FPGA overcomes the problem of sequential execution incurred by a processor, the proposed design used a direct approach to convert LD into Verilog code.

2. Ladder diagram to HDL code – A direct mapping

PLC manufacturers define their own conventions for the elements to be represented in LD. Therefore, representation of inputs, outputs, addresses and other functional blocks differs with respect to the manufacturers [12]. This requires the programmers to pay attention to the

naming followed for a particular PLC irrespective of their programming skill. LD has many function blocks and the basic elements called Normally Opened (NO), Normally Closed (NC), output and timer coils. The logic of entire LD is mainly decided based on the connections established by the basic elements. Hence this work has focused on the basic elements. Solution for the conversion of LD into HDL code has been developed using data flow graph, state chart [13 – 14]. This work has concentrated on how to directly generate HDL code from the LD for each rung.

The process of LD development to FPGA implementation is done in two stages. In the first stage, Laboratory Virtual Instruments for Engineering Workbench (LabVIEW) application software generates an equivalent VerilogHDL code for the LD developed in LabVIEW. Thus generated code can be implemented into an FPGA. Suitable synthesis software is used in the second stage to convert VerilogHDL code into a bit file. Synthesis software is normally provided by FPGA vendor. Therefore, this work has focused its attention on the synthesis process of LD to VerilogHDL code.

Fig. 3 depicts the proposed novel idea of converting LD into VerilogHDL code. Each rung in an LD is represented using '*assign*' statement. The nature of '*assign*' statement in VerilogHDL language is its concurrent execution. Therefore, this method overcomes the problem of sequential execution of LD in a rung by rung basis by a processor. Again, elements mentioned in a rung are also executed sequentially. However, if any one of the variable in an '*assign*' statement depends on the output of another '*assign*' statement then the concept of concurrent execution would become invalid.

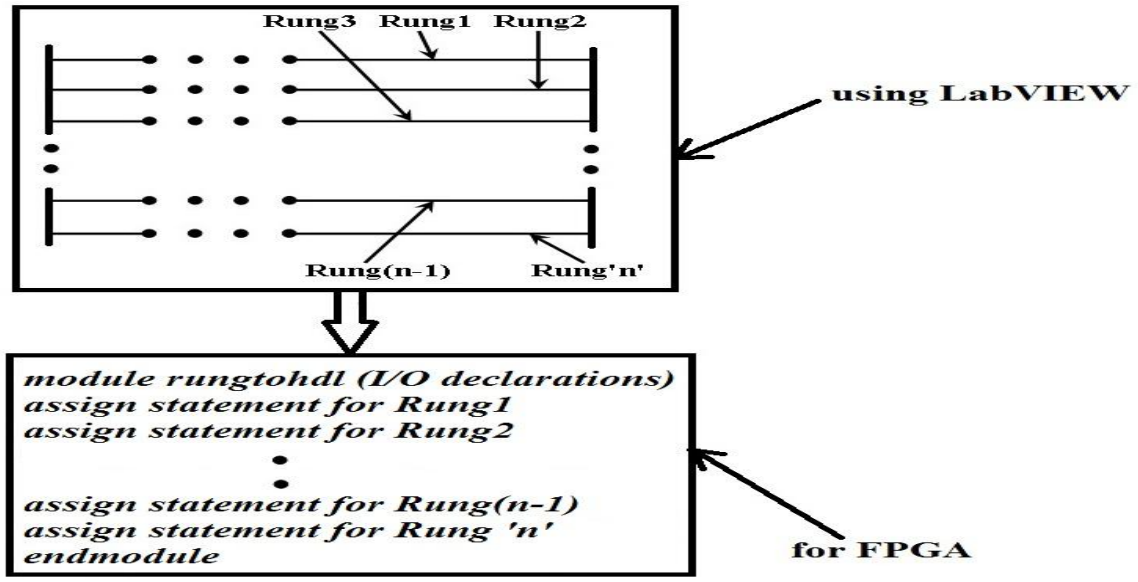


Fig. 3 Proposed idea of converting LD into VerilogHDL

Synthesis of 'assign' statement results into combinational circuits. Use of timer element in the LD demands clock signals and this creates sequential circuits. LD elements considered in the synthesis process are NO coil, NC coil and timer [15]. Three timers considered for the purpose of synthesis in this work are pulse timer, on delay timer and off delay timer. LabVIEW invokes the pre-written VerilogHDL code whenever a particular type of timer is included in the LD. A complete VerilogHDL code for the entire LD will be generated when the "generate code" option in the application software is chosen. Fig. 4 shows the VerilogHDL code of the logic gates equivalent to the LD representation that has used "assign" statement. This has been the basic synthesis model of LD to VerilogHDL code conversion of this work.

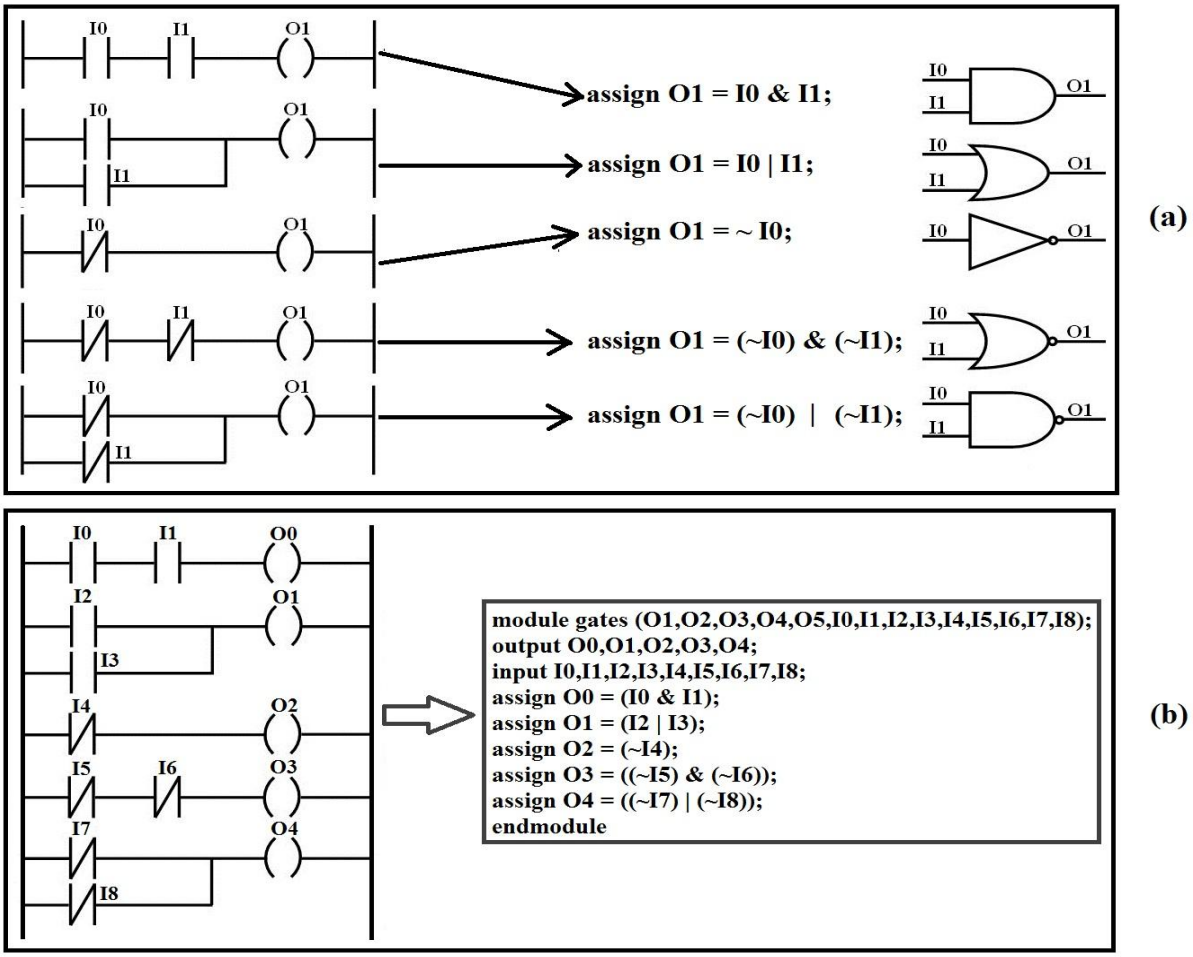


Fig. 4 Ladder diagram, logic gate and VerilogHDL Description

'AND' logic gate is realized when two 'NO' coils are connected in series. It becomes a logical 'OR' gate when these two coils are connected in parallel. Similarly, 'NOR' and 'NAND' gates are realized by connecting 'NC' coils in series and parallel respectively. VerilogHDL code

shown in Fig. 4 will be packed into a module when all these rungs are programmed into an LD to ensure concurrent execution.

An example of LD and its equivalent digital circuit after synthesis are shown in Fig. 5. The normal way to execute O1 is to store O0 in a memory element and use this data whenever the second rung is under execution. Memory element requires a clock pulse and also creates delay. On the other way, the equivalent VerilogHDL code for this LD is generated and synthesized to generated RTL circuit as in Fig. 5.

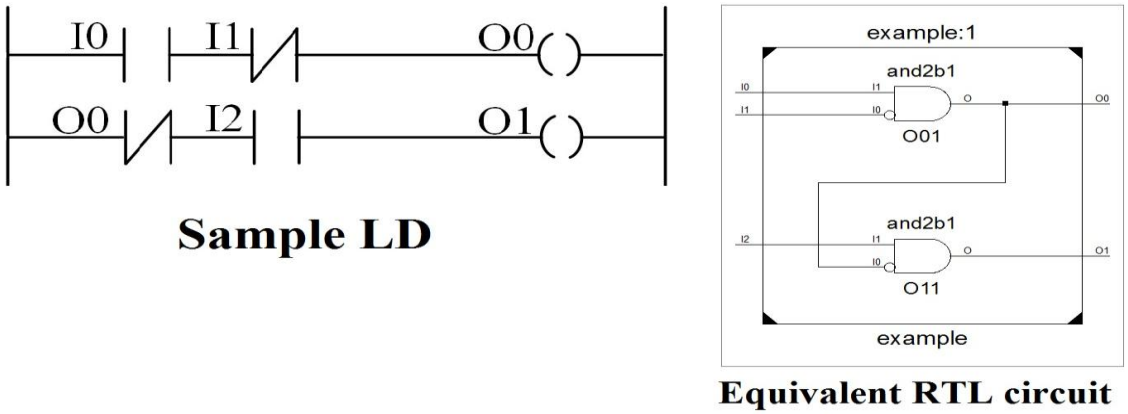


Fig. 5. Sample LD and its synthesized digital circuit

The Boolean expression for the LD can be derived as below.

$$O0 = I0 \,\&\, (\sim I1) \tag{1}$$

$$O1 = (\sim O0) \,\&\, I2 \tag{2}$$

$$O1 = I0 \,\&\, (\sim I1) \,\&\, I2 \tag{3}$$

Eq. (3) avoids the usage of memory element as O0 is directly connected to the second rung. This facilitates merging of two rungs into a single. Therefore, the synthesis software is intelligent enough to generate an optimized Register Transfer Logic (RTL) circuit, utilize the FPGA resources in an efficient way. Hence, it is enough to generate the equivalent HDL code for the given LD. In LDs, outputs are sometimes used as inputs. Therefore, declaring the output as bi-directional in VerilogHDL code, it can be connected as an input to another logic gate. This work has followed three steps to generate VerilogHDL code from the LD.

- Develop LD in the front panel of LabVIEW
- Generate Verilog code for the present rung
- Generate Verilog code for all the rungs

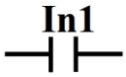
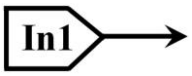
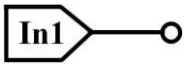
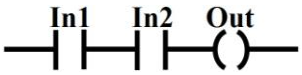
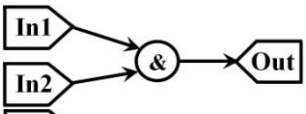
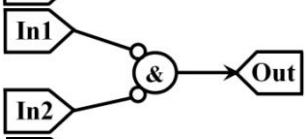
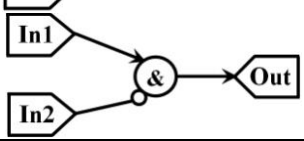
2.1 LD elements description in LabVIEW

Basically, LabVIEW software was developed for virtual instrumentation. Apart from instrumentation, it can be used in many broader areas like signal processing, communication, image processing, control design, simulation, data analysis etc., It offers a graphical programming environment. Programming can be done either at low level or functional level. This work has used LabVIEW to develop LD in its front panel and generate equivalent VerilogHDL code from it. This work has mainly used the functions of Event and Case structure. Rungs in an LD are executed in a series manner. Efficiency of execution can be improved by rearranging them without disturbing the logic involved in it. Representation of LD using enhanced data flow graph (EDFG) has been discussed in [15]. This work has involved EDFG to represent LD elements as and when it is required.

2.1.1 NO / NC – Series

Output of an NO coil follows its input. In case of NC coil, the output is the inverse of its input. It is possible that a rung can have more than one NO coils connected in series or NC coils connected in series or the combination of both NO and NC coils connected in series. Table 2 indicates the representation of NO and NC coils and the possibilities by which these two can be connected in series using EDFG. Variable names assigned for the inputs and outputs and the functions of the LD elements discussed in this work are based on the specifications mentioned by the IEC6113-3 standard [16].

Table 2: EDFG Representation of NO and NC coil connected in series

Sl. No	Description of LD element(s)	Representation in a rung	Representation using EDFG
1	NO Coil		
2	NC Coil		
3	Two NO coils connected in series		
4	Two NC coils connected in series		
5	NO and NC coil connected in series		

Respective equivalent VerilogHDL codes for the Sl.No 3, 4, and 5 are listed below.

assign out = in1 & in2;

assign out = (~in1) & (~in2);

assign out = in1 & (~in2);

In LabVIEW, Series event structure is invoked when the user clicks on the "SERIES" control button. This structure is used to draw the symbol of NO / NC coil. Fig. 6. shows the block diagram (coding part) for series or parallel connection of elements in LD. Case structure inside this event structure will check for the coil type (NO / NC). After choosing the coil, this particular coil will be included in LD; name assigned for this will be stored in memory. This event structure can be called 'n' number of times to connect 'n' elements in series. However, this is to be limited in the practical scenario.

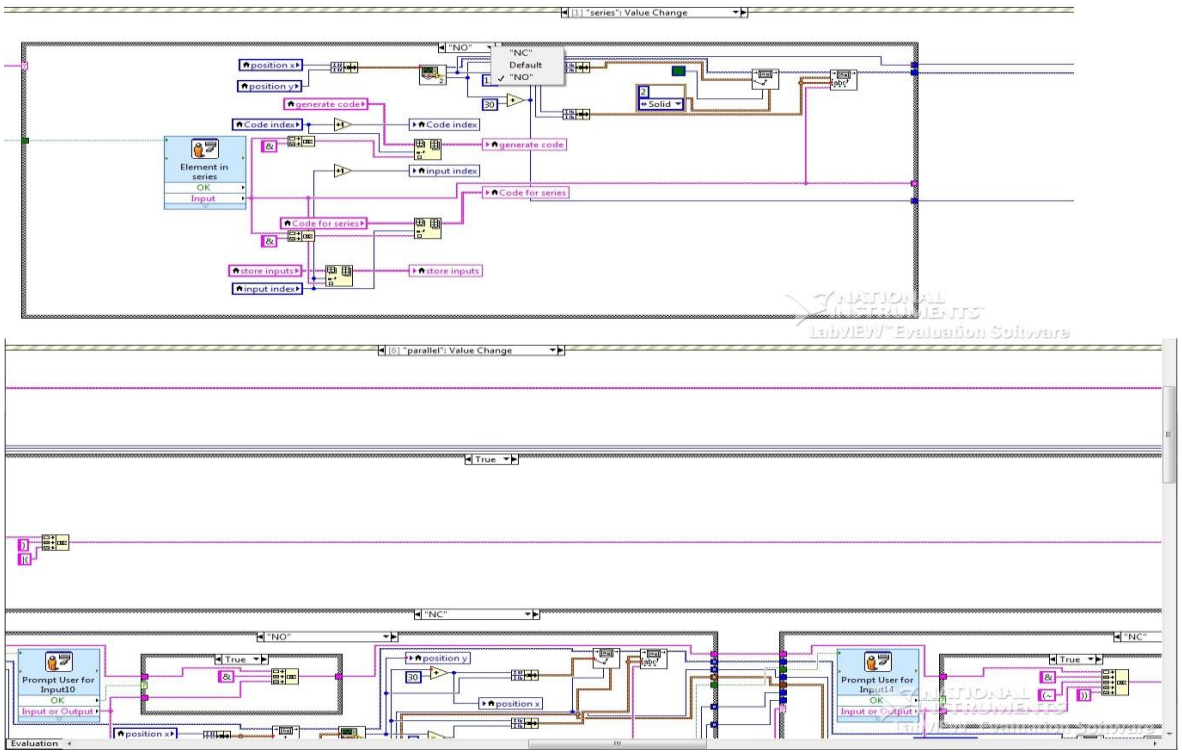
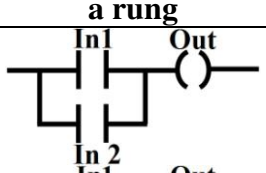
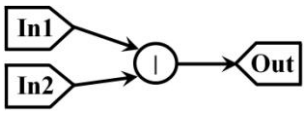
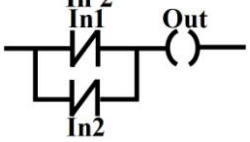
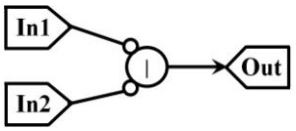
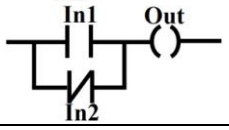
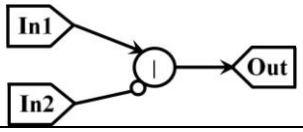


Fig. 6. Event, case structure for series & parallel connection

2.1.2 NO / NC – Parallel

It is possible that NO and NC coil can be connected in parallel. Table 3 explains how these connections can be represented using EDFG. It shall be noted that the representation has been explained for only one NO and NC coil. However, in practice, more number of NO / NC coils are used in the LD.

Table 3: EDFG Representation of NO and NC coil connected in parallel

Sl. No	Description of LD element(s)	Representation in a rung	Representation using EDFG
1	Two NO coils connected in parallel		
2	Two NC coils connected in parallel		
3	NO and NC coil connected in parallel		

Respective equivalent VerilogHDL code for the Sl.No 1, 2, and 3 are listed below.

assign out = in1 | in2;

assign out = (~in1) | (~in2);

assign out = in1 | (~in2);

Fig. 7 shows a rung in which an NC coil (In4) is connected in parallel with the three NO coils (In1, In2, In3) that are connected in series. It also indicates how this rung is denoted using EDFG.

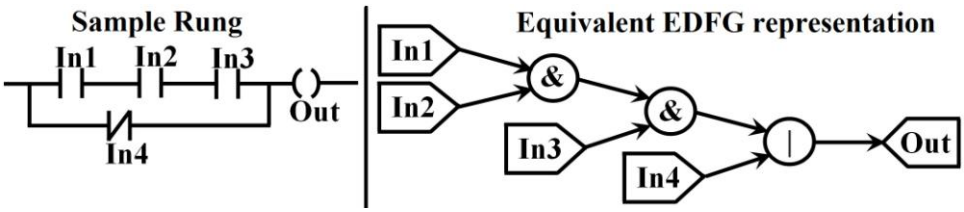


Fig. 7 Sample rung and the EDFG representation

Since the NO coils are logically ANDed with each other, the EDFG can be simplified as shown in the Fig. 8. Also, this rung can be represented in VerilogHDL code using "*assign*" statement as mentioned below.

assign out = ((in1 & in2 & in3) | (~in4));

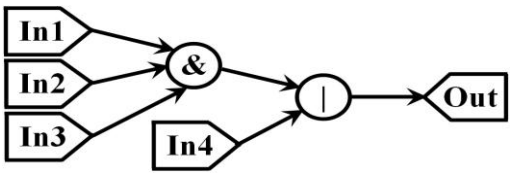


Fig. 8 Simplified EDFG representation

Application software invokes the parallel event structure as and when the user clicks on the "PARALLEL" control button. In parallel connection, more than one element can be connected in series. Maximum number of elements in series connection must be decided during the design time itself. User has to assign the total number of elements in series, type of the coil and the corresponding name. 'n' case structure can be cascaded for 'n' elements in series. Similarly, successive parallel connections can also be established. Names assigned to each element will be store in memory. Every rung must be ended with an output. Hence VerilogHDL code for the present rung will be generated whenever the output event structure is called. Thus generated code will be stored in memory. Code for the entire LD will be generated when the generate code event structure is called.

2.1.3 Timer

IEC61131-3 standard defines three types of timers namely, Pulse Timer, On-delay Timer and Off-delay Timer [16]. These timers have the input variables IN, Pre-set value (PT), output variables Q and End Time output (ET). Data type of the parameters IN and Q are Boolean. Status of Q is changed based on the Pre-set value of the timer. Delay created by the timer can be read from ET. Timing diagrams of all the timers mentioned by the IEC61131-3 has been produced here to have better understanding of the function of timers [16].

2.1.3.1 Pulse Timer

Pulse timer activates the timer to start counting whenever it detects rising edge of the input IN. Clock pulse of FPGA is used as a pulse generator to the timer. Output Q is set until the count value becomes equal to the value of PT. It is reset once the counter reaches the value of PT and the counting is also stopped. Timer count value is updated to the output ET. It continues to count even when the status of IN is changed to falling edge before the value of PT is reached. Fig. 9 shows the timing diagram of a pulse timer.

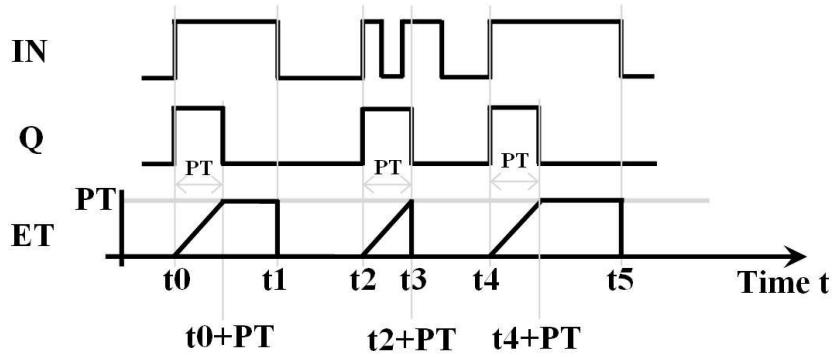


Fig. 9 Timing diagram of Pulse Timer

Description of pulse timer in the form of EDFG is shown in Fig. 10. Rising edge of the input IN activates 'start'. Start is used as s register. This will be reset only when the value of 'tcount' exceeds PT.

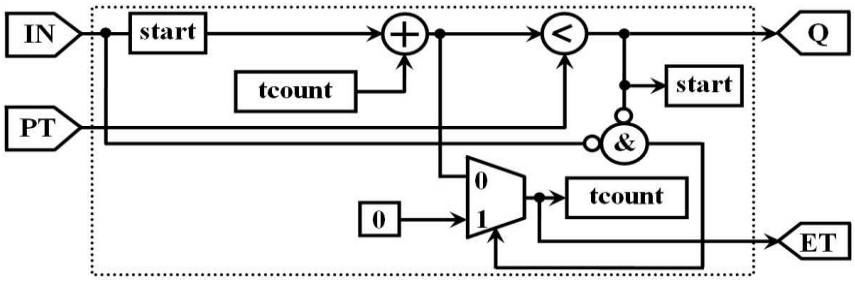


Fig. 10 EDFG representation of pulse timer

'tcount' is used as a counter register. It is reset only when its value is greater than PT and the IN value is low.

2.1.3.2 On delay Timer

Output Q of on delay timer follows its input after the preset time delay is reached. Timer counting is in active condition till the input IN is high. When the IN status is changed to low, timer counting is reset to zero. Present time delay of the timer can be identified by referring the output ET. Timing diagram of on delay timer as specified by the IEC61131-3 standard is shown in the Fig. 11.

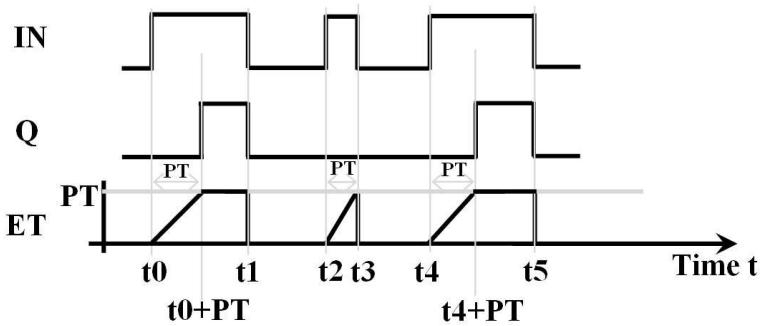


Fig. 11 Timing diagram of on delay Timer

'tcount' shown in Fig. 12 is used as a register. This register is updated under two conditions. When the input of IN is low, its value is reset to zero. Similarly, incremented count value for every clock pulse is assigned to this register. Also, register value is assigned to the output ET as this is the requirement of IEC61131-3. This value can be used during the time of programming LD.

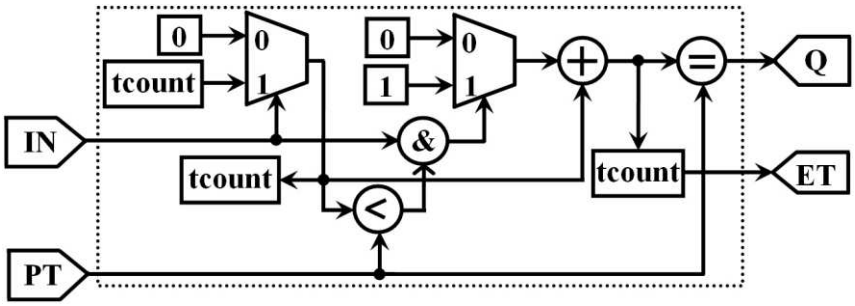


Fig. 12 EDFG representation of on delay timer

2.1.3.3 Off delay Timer

Function of off delay timer is the inverse function of on delay timer. It is activated for counting whenever it identifies a falling edge of the input IN. Output of Q is high when the input of IN is high. Q is changed to low when the timer count value reaches the preset value PT. This status is retained till the time the input to IN is low. Timer is reset when the IN becomes high. As shown in the Fig. 13, if the IN is changed from low to high before the count value reaches PT, timer will be reset.

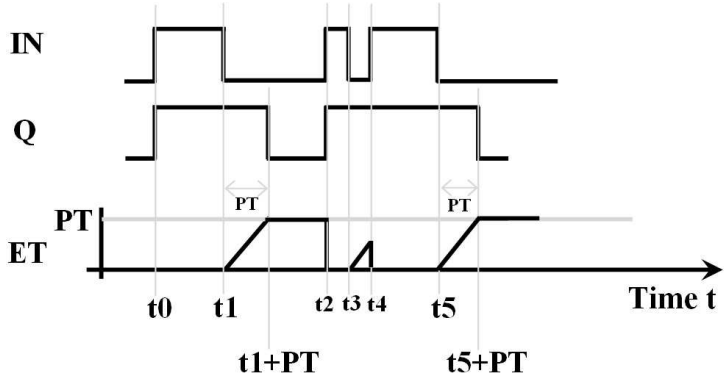


Fig. 13 Timing diagram of off delay Timer

EDFG representation for the off delay timer is similar to the on delay timer. Changes are made in the logical AND operation and the connection to the Q as shown in the Fig. 14. IN input is connected to the AND operation after it is inverted. Similarly, the comparison output is also inverted before connecting to Q. This ensures that the diagram satisfies the functionality of off delay timer.

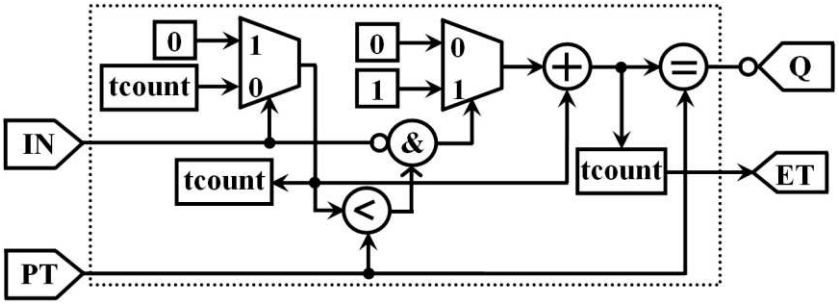


Fig. 14 EDFG representation of off delay Timer

LabVIEW invokes timer event structure whenever the "TIMER" control button is clicked. User can choose any one of three timers. By selecting any type of timer, user has to enter the value of preset time. Clock frequency of the FPGA will be known only at the time of implementing the design into an FPGA. Therefore, the user has to identify the clock frequency of FPGA while developing the LD and calculate the count value required to create delay in terms of nano seconds. This count value must be entered whenever timer is used in the design to create delay in the operation. Thus the delay time is accepted in the units of nano seconds by the application software. Timer event structure is responsible to draw the symbol of timer and assigns suitable name to it. The application software moves to the next rung after drawing the symbol of timer. Timer output is assigned to a register variable. Programmer can use this variable in LD to create a suitable logic. Application software has used 'x' as the timer variable. LD can have more than one timer. Hence a numeric index is concatenated with the variable 'x'. For example, if there are two timers in the LD, then the first timer will be assigned as 'x0' and the second timer will be assigned as 'x1'.

3. Application software design, VerilogHDL code generation

Application software design was divided into two parts. One part is responsible to create a graphical display of LD. It also displays the names given to a particular element and stores them in an array. Second part of the software uses this array content to generate VerilogHDL code. Application software generates the code as and when an LD element is placed and stored in an array. It expects the user to do the below actions.

- Choose the element – NO / NC / Timer
- Connection of NO / NC coil – Series / Parallel
- Assign a suitable name to the coil

Event structure function waits for an event to happen. List of events included in the software are,

- New
- Series
- Parallel
- Timer
- Output
- Generate code
- Stop

Upon the selection of "New" control button, the content of 2D picture indicator will be deleted. A horizontal line will be drawn to indicate the beginning of a first rung. "Series", "Parallel" control buttons are used to place the selected element (NO / NC) either in series or parallel connection respectively. User is requested to assign a suitable name corresponding to the chosen element. "Timer" control button shall be invoked to include the timer in the LD. Time delay must be entered in the unit of nano seconds. Every rung must be ended with an output. Therefore, a rung will be completed only when the ""Output" control button is chosen.

Upon the "Output" selection, symbol for output will be drawn and the next rung will be started. LabVIEW code to draw NO, NC, output and timer elements have been defined as a sub vi. Particular sub vi will be called to draw the symbol based on selection. "Generate code" control button is used to generate Verilog HDL code for the LD developed by the user. This work has utilized RTL level descriptions, especially "assign" instruction. Statement content is generated in association with the LD drawn by the user.

3.1 Series connection – NO / NC Coil

A case structure function inside "series" event structure function will generate VerilogHDL code for this condition. Since the output of NO coil follows its input, name assigned to it is entered as an input variable. Series connection represents logical AND operation. Hence, the symbol for AND operation in VerilogHDL code "&" is added with the element's name. If the coil type is NC, then the input must be inverted. Therefore, symbol for invert operation in VerilogHDL code "~" and "&" symbol for AND operation are added before and after the name respectively. Thus generated part of the statement is stored in an array.

3.2 Parallel connection

A cascaded case structure function inside "parallel" event structure function will generate VerilogHDL code for the elements connected in parallel. NO / NC coil shall be connected in series in the parallel connection. This application has limited the maximum number of elements that can be connected to three (In the combination of NO / NC). Dialog window displayed expects the user to fill the details of starting and end point of parallel connection and the NO / NC specification. Number of elements connected in series can be further extended. One case structure function must be cascaded for every addition of an element. Parallel connection indicates the logical OR operation. Symbol for OR operation in VerilogHDL code is "|". This will be included whenever the parallel event structure is invoked. "&" symbol will be included for every element connected in series.

3.3 Timer

Timers that shall be included in the LD are pulse timer, on delay timer and off delay timer [16]. Application software is able to generate Verilog HDL description for these timers.

3.3.1 Pulse timer

Functionality of pulse timer has been already explained in the section 2.1.3.1. Based on the requirements of IEC 61131-3, Verilog HDL code that will be generated by the application software for this timer is listed below. Input and output variables in, pt, q, et declared in this design are similar to those variables declared by IEC61131-3. This same convention is followed for the remaining timers also.

```
always @(posedge clk)
begin
    if (in == 1)
        start = 1;
    if (start == 1 && (count <= pt))
    begin
        count = count + 1;et = count;q = 1;
    end
    if (count > pt)
    begin
        q = 0;count = 0;start=0;et=0;
    end
```

```
end
```

User enters the delay time to the input variable PT. 'count' is declared as a 32-bit counter with an initial value of zero. Separate hardware must be assigned for each timer to ensure concurrent execution. Hence, the timer index value will be concatenated with the timer counter register 'tcount' and the timer output register 'x'. Variable 'in' triggers the single bit register 'start' to ensure that the timer is activated. 's' register is reset as and when the input of 'in' goes low.

3.3.2 On delay timer

On delay timer is started whenever its input is "high". If the 'in' input is changed from the state of "high" to "low" then the timer value will be reset to zero. Timer output will be changed from the state of "low" to "high" only when the timer value becomes equal to the value set by the user. Counter value is reset whenever 'in' input is low. Application software generates the below Verilog HDL code when this on-delay timer is used in LD.

```
always @(posedge clk)
begin
        if (in == 1 && count < pt)
        begin
                count = count + 1;et = count;
                if (count == pt)
                        q = 1;
        end
        if (in == 0)
                begin
                        q = 0;count=0;et = 0;
                end
end
```

3.3.3 Off delay timer

Output of off delay timer is the inverse of its input after the timer reaches the time delay specified by the user. It will be started only when the 'in' input is low. Thus the default status of timer output 'q' is defined to be high. It is changed to low when the count value 'count' reaches the value of 'pt'. VerilogHDL code generated by the application software for this type of timer is shown below.

```
always @(posedge clk)
begin
if (in == 0 && count < pt)
        begin
                count = count + 1;et = count;
                if (count == pt)
                        q = 0;
        end
        if (in == 1)
                begin
                        q = 1;count=0;et = 0;
                end
end
```

3.4 Output coil

Selecting an output element indicates the end of rung. Therefore the application software draws the symbol for output and moves the cursor for next rung. It also generates VerilogHDL code as and when one rung is completed. Actually, LD allows an output to use as an input in any rung. "inout" allows a defined variable to be used as either input or output. Hence, by default, all the outputs are declared as "inout". Based on the logic, same input or output may be used in more than one rung. This creates the duplication of input or output during VerilogHDL code generation. Multiple declaration leads to synthesis error. Hence, the multiple usages are identified and only one variable is stored in memory for VerilogHDL code generation. After the generation of VerilogHDL code for the current rung, all the memory elements are initialized to default value.

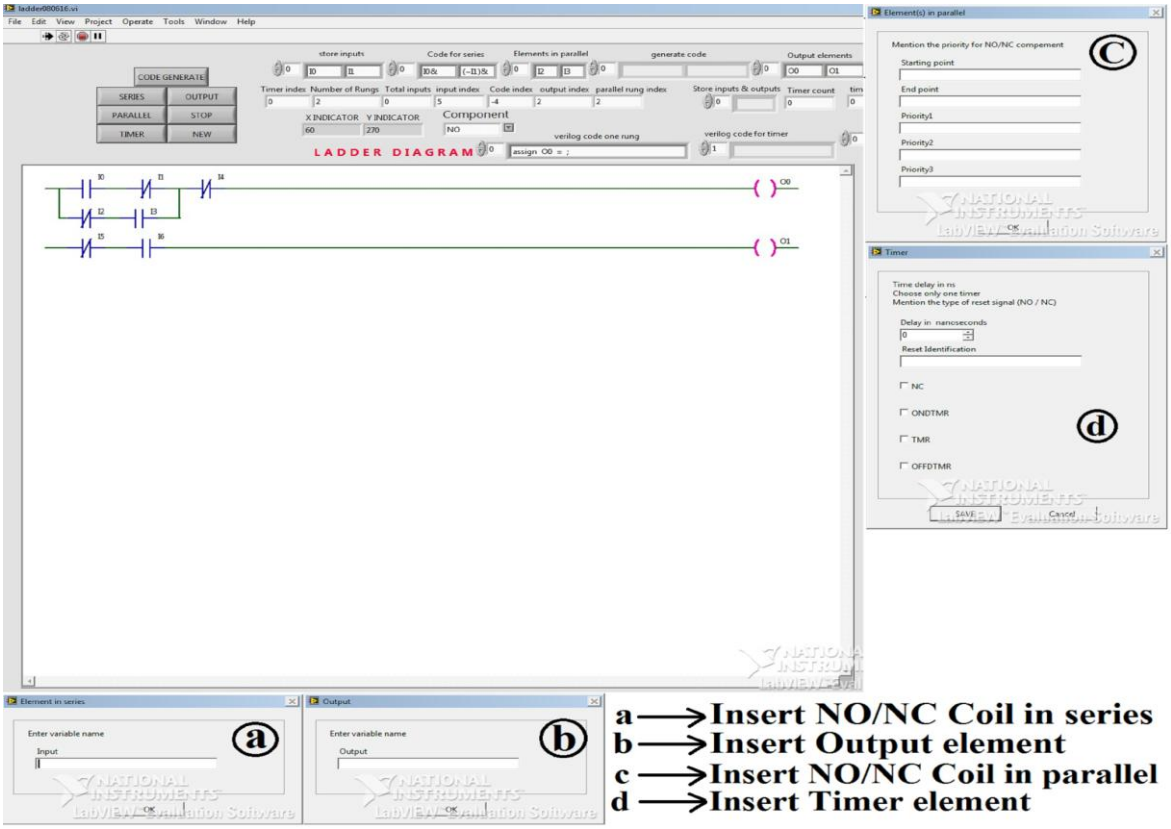


Fig. 15 Application software front panel, dialog boxes

Complete LD developed using application software can be viewed in the front panel as shown in Fig. 15. Various dialog boxes invoked by the click of control buttons are also shown. VerilogHDL code for the entire LD is developed when the "code generate" event structure is chosen as listed below.

- Form the module name
- Retrieve all the inputs and outputs
- Declare inputs and outputs (all the outputs will be declared as inout)
- Read the individual rungs and concatenate
- Retrieve the VerilogHDL code generated for timer(s)
- Mention the "endmodule" statement
- Concatenate all the lines developed using the above steps
- Store the VerilogHDL code in a text file

Now this file can be copied and used for synthesis and FPGA implementation.

4. Case study and FPGA implementation

Two simple case studies have been presented to verify the functionalities of the application software. They are,

- Substances mixing
- Traffic light control system

Case study of substances mixing has not used any of the output elements as inputs. Thus the design of LD is converted into a combinational circuit. Design of traffic light controller has used both the on-delay timer and off-delay timer. Also, the outputs of one signal are used as an input for the other. Thus the design of LD for traffic light controller has used sequential circuit.

4.1 Mixing of substances

Let an automated process in which water is mixed with two or more substances. Process tank has two level sensor limit switches fixed. One is fixed at the bottom of the tank to indicate low water level (LLLS) while the other is fixed at the top of the tank to indicate high water level (HLLS). Whenever, LLLS output is low, control valve (CV1) opens and hence water enters into the tank. At the same time, a motor is also switched ON to mix the substances with water. This process continues till HLLS output becomes high. A high output in HLLS closes CV1 and switches OFF the motor.

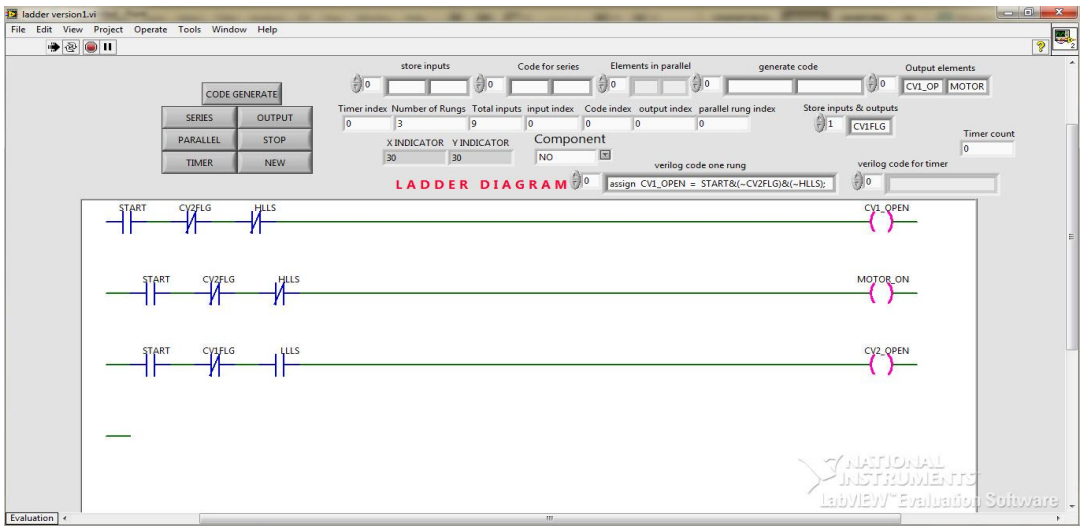
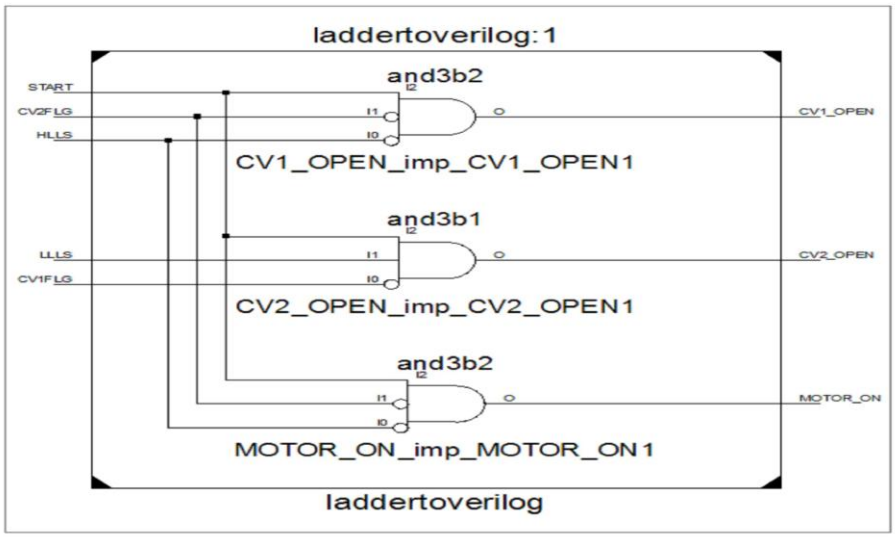


Fig. 16 Ladder diagram developed using the application software for substances mixing

Now, control valve (CV2) at the bottom of the tank is opened to empty the tank. It will be closed once the output of LLLS is low. Above mentioned problem is drawn as LD as in Fig. 16. VerilogHDL code was generated by the activation of "generate code" control

button click. Thus generated code was stored as a text file in a location as mentioned in the program.




```
module laddertoverilog (CV1_OPEN,MOTOR_ON,CV2
_OPEN,START,CV1FLG,LLLS,CV2FLG,HLLS);
inout CV1_OPEN,MOTOR_ON,CV2_OPEN;
input START,CV1FLG,LLLS,CV2FLG,HLLS;
assign CV1_OPEN = START&(~CV2FLG)&(~HLLS);
assign MOTOR_ON = START&(~CV2FLG)&(~HLLS);
assign CV2_OPEN = START&(~CV1FLG)&LLLS;
endmodule
```

Fig. 17 RTL Schematic showing three AND gates for the three assignment statements

Fig. 17 shows the RTL schematic of the code generated during synthesis. Unlike LD execution by a PLC, RTL logic generates output as and when there are changes in the inputs. Also, RTL schematic shows individual logic gates for each rung. At this stage, there is no logic optimization. In synthesis process, Technological schematic will be generated after RTL schematic generation. Synthesis software, generates the required hardware resources after optimizing the RTL logic as in Fig. 18.

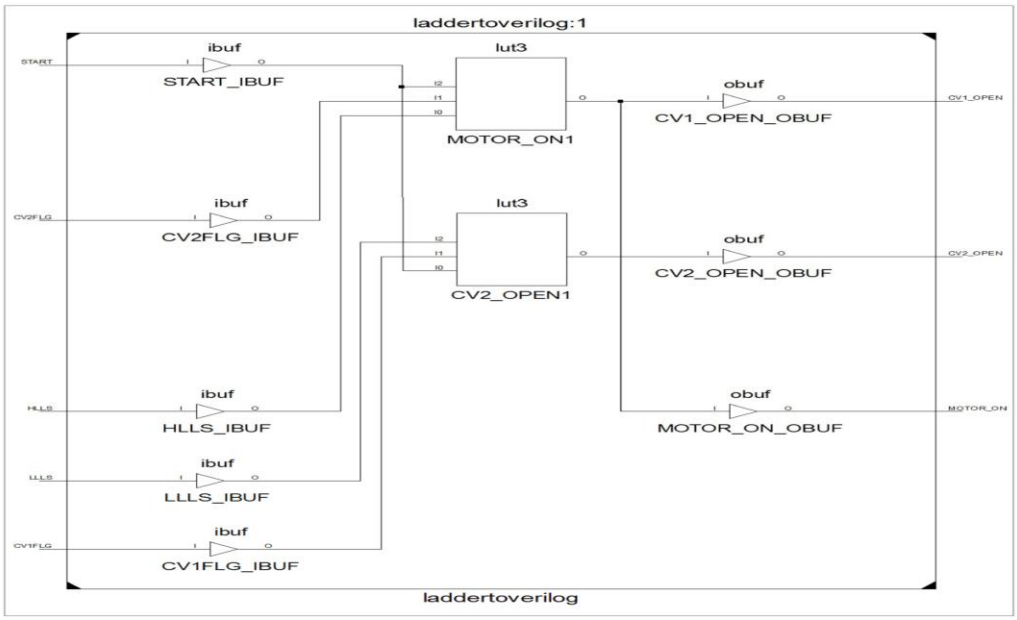


Fig. 18 Technological schematic after synthesize for hardware optimization

Rung 1 and rung 2 in LD are similar except that the outputs are different. Even though RTL schematic assigned two separate logic gates to perform similar functionalities, synthesize software is intelligent enough to identify this and assign two different outputs from the same hardware. Hence the programmer does not need to worry about logic and hardware optimization.

4.2 Traffic light control system

Design of traffic light controller has been one of the popular applications. This work has considered this application to prove that the application software is able to generate VerilogHDL code for sequential circuits also. LD was developed for two alternate sequences of traffic light control. In the first sequence, traffic is allowed to in the directions of north and south. At the same time, traffic in the directions of east and west are stopped. Now, the traffic conditions are reversed in the second sequence. The sequence is again started from the first sequence. Fig. 19 shows the LD developed using the application software for traffic light control system. Switching sequence of light is as below.

- Red colour light to yellow colour light
- Yellow colour light to green colour light
- Green colour light to red colour light

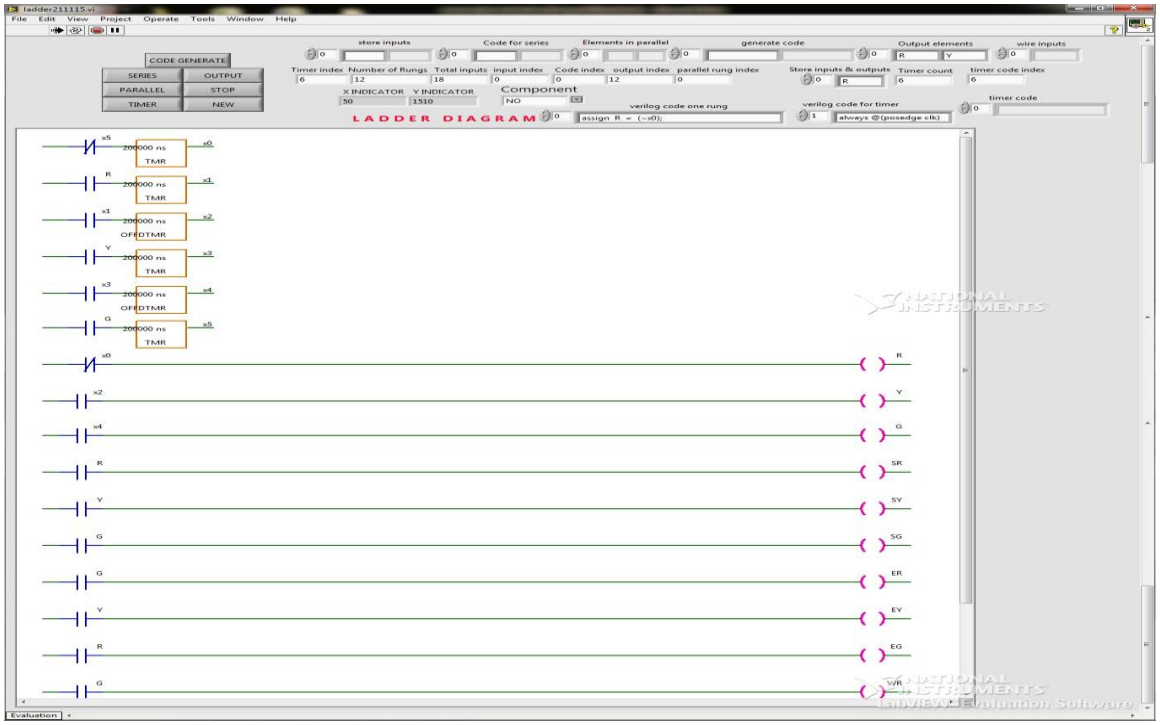


Fig. 19 Ladder diagram for traffic light control system

Delay value between the sequences is entered as 2,00,000 to create the delay of 4 seconds.

4.3 FPGA implementation

It is easy to implement the VerilogHDL code into FPGA by following a set of predefined procedures. It demands the conversion of VerilogHDL code into a bit file. It is compulsory to assign each NO / NC coil in the LD to FPGA pins as an input or output. In other words, VerilogHDL code expects that the user assigns inputs and outputs to FPGAs to the variables declared in the module. Details of the pin assignment are available in user constraint file (UCF). FPGA vendor assigns suitable names to FPGA pins and by referring these details, one can assign variables to FPGA pins. Synthesis software uses VerilogHDL code and UCF file to generate FPGA downloadable file in the bit format. Joint test action group (JTAG) interface between the computer and FPGA board helps to download the bit file into FPGA. After downloading, FPGA acts as a standalone device. Table 4 lists the inputs and outputs required for this problem.

Table 4: Descriptions and functions of inputs and outputs for mixing of substances

Sl.No	Item description	Input / Output	Function
1	Start button	Input	Controls the process.
2	Low level limit switch	Input	Limit switch will be ON whenever the tank level is low.
3	High level limit switch	Input	Limit switch will be ON whenever the tank level is high.
4	Control valve1 flag	Input	Status of control valve1 identified through a limit switch in it.
5	Control valve2 flag	Input	Status of control valve2 identified through a limit switch in it.
6	Motor	Output	Will be started whenever the tank level is low.
7	Control valve1	Output	Will be opened whenever the tank level is low and closed when it is high.
8	Control valve2	Output	Will be opened when the tank level is high and closed when it is low.

5. Data acquisition system

In industries, a computer in a centralized place gets analog and digital signals from the field instruments. Software in that computer reads the signals and displays the actual process condition. This facility is necessary to ensure safety in process, quality product. It is possible to generate control signals from the computer and changes the process condition. Such a kind of set up is known data acquisition system (DAS). In industrial automation, PLC acts as a mediator between computer and field instruments [18]. It takes a specific time period for data transfer. PLC scan time is inclusive of this time also.

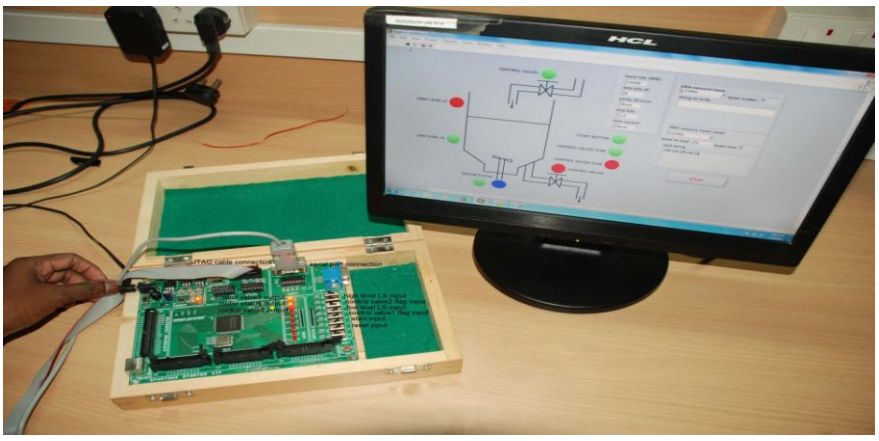


Fig. 20 FPGA implementation of case study and its data acquisition system

Actually, this is an extra load to PLC. In the proposed design, a portion of FPGA hardware resources takes care of data transfer between FPGA and computer. FPGA is responsible to

acquire signals from the field. This guarantees independent operation of logic execution of acquired signals from the field and data transfer to the computer. Fig. 20 shows FPGA implementation of case study for mixing of substances. DAS program in the computer displays details of inputs and outputs. RS 232 communication establishes serial data transfer between FPGA and computer. Many process industries and research laboratories use LabVIEW to develop DAS system [14]. Fig. 21 shows LabVIEW code for both serial data communication and DAS system.

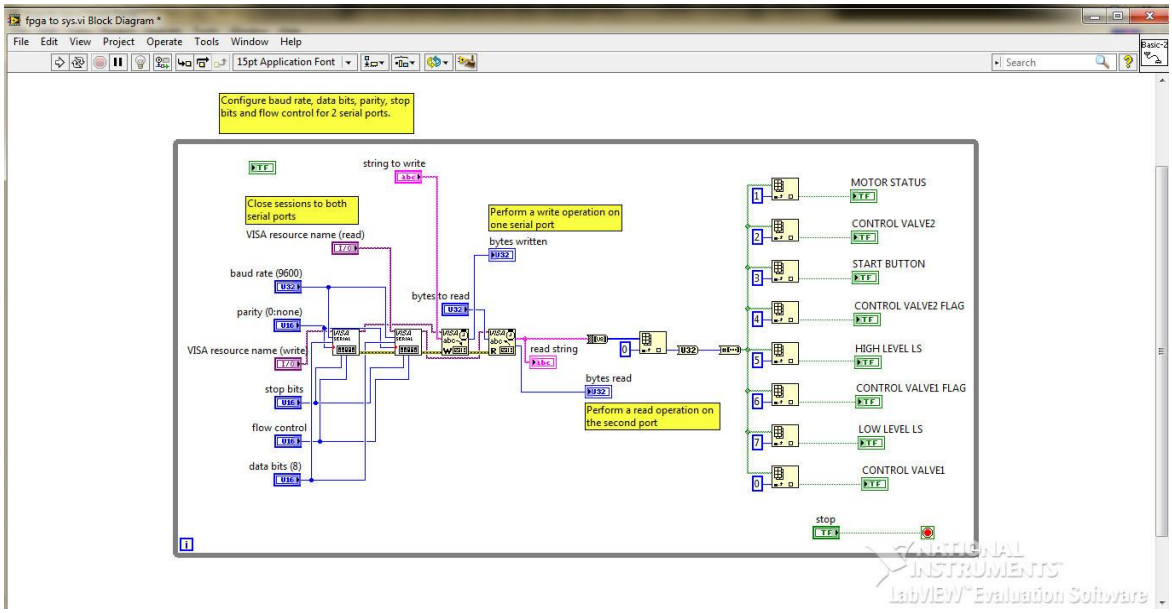


Fig. 21 LabVIEW code for serial communication and display data received from FPGA

Following parameters must be similar on both the sides of a system to establish serial data transfer.

- Baud rate
- Data bits
- Parity bits
- Stop bits
- Number of bytes to read / write

Speed of data transfer depends on the baud rate. It will be more when the baud rate is high. However, establishment of high speed communication reduces cable length. LabVIEW software was used on the system end to receive data from FPGA. It reads data sent by FPGA as a string. String to byte array function converts string data into Boolean.

5.1 Serial communication - FPGA side

VerilogHDL code for serial communication involves baud rate generation, converting parallel data into serial using shift register and start sending the serial bits one by one till transmit bit is set. Baud rate represents the number of bits sent per second. There is a relationship between baud rate and FPGA clock pulse. Dividing FPGA clock pulses for a predetermined count decides the baud rate. Therefore, a comparator compares the counter value with predetermined number of clock pulses.

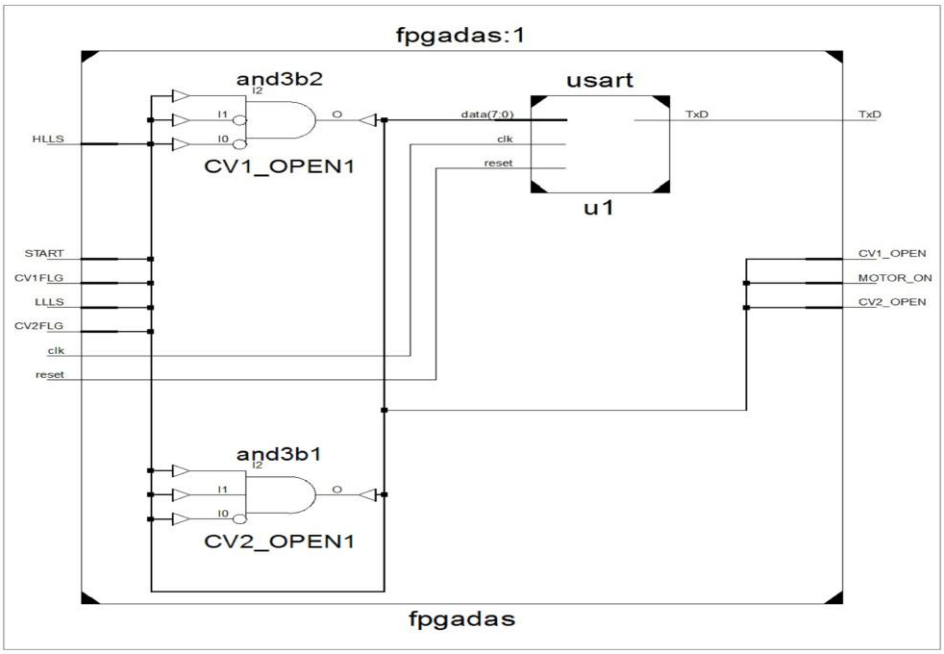


Fig. 22 Technological schematic that has separate hardware for logic execution and serial data transfer

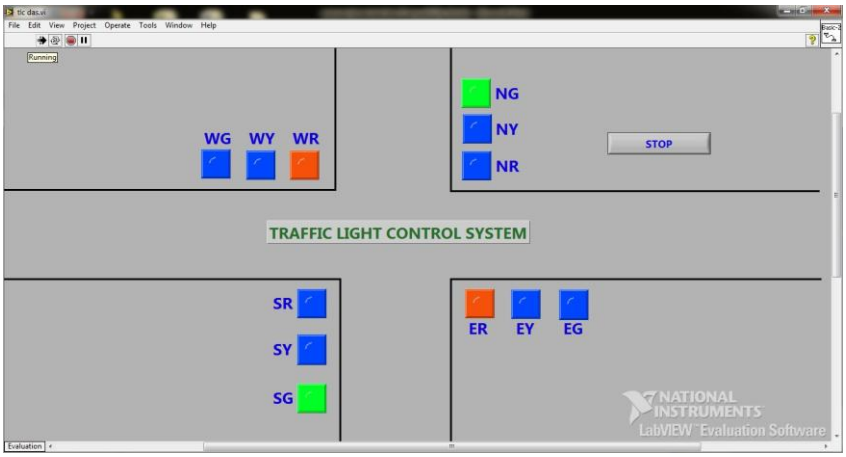


Fig. 23 LabVIEW display of traffic light control system

Counter increases its count for every clock pulse. Since the computer does not generate any control signals to FPGA, VerilogHDL code was generated for transmitter alone.

RTL schematic of DAS in Fig. 22 shows that FPGA allocates separate hardware for logic execution and serial communication. Fig. 23 shows DAS system of traffic light control system.

6. Results & Discussions

Main focus of this work is to develop a feasible solution for the generation of VerilogHDL code equivalent to the LD. In addition to that, it also focused on the following functionalities to verify the design and provide a diagnosis system.

- Case study for the conversion and its FPGA implementation
- Data acquisition system of FPGA

Table 5: Xilinx FPGAs requiring hardware for logic execution and serial data transfer for both the case studies

Target Device	Device family	Logic utilization	Used	Available	Utilization
		Case study – Mixing of substances			
XC3S200	SPARTAN 3	Number of slices	26	1,920	1%
		Number of slice flip flops	29	3,840	0.007%
		Number of 4 input LUTs	47	3,840	1%
		Number of bonded IOBs	11	97	11%
		Number of GCLKs	1	8	12%
XC6SLX4	SPARTAN 6	Number of slice registers	27	4,800	0.005%
		Number of slice LUTs	73	2,400	3%
		Number of fully used LUT-FF pair	27	73	36%
		Number of bonded IOBs	11	102	10%
		Number of BUFG/BUFGCTRLs	1	16	6%
XC5VLX330T	VIRTEX 5	Number of slice registers	29	2,07,360	0%
		Number of slice LUTs	65	2,07,360	0%
		Number of fully used LUT-FF pair	29	65	44%
		Number of bonded IOBs	11	960	1%
		Number of BUFG/BUFGCTRLs	1	32	3%
		Case study – Traffic light control system			
XC3S200	SPARTAN 3	Number of slices	248	1,920	12%
		Number of slice flip flops	178	3,840	4%
		Number of 4 input LUTs	412	3,840	10%
		Number of bonded IOBs	14	97	14%
		Number of GCLKs	1	8	12%
XC6SLX4	SPARTAN 6	Number of slice registers	178	4,800	3%
		Number of slice LUTs	516	2,400	21%
		Number of fully used LUT-FF pair	177	517	34%
		Number of bonded IOBs	14	102	13%
		Number of BUFG/BUFGCTRLs	1	16	6%
XC5VLX330T	VIRTEX 5	Number of slice registers	178	2,07,360	0.001%

Number of slice LUTs	386	2,07,360	0.002%
Number of fully used LUT-FF pair	173	499	44%
Number of bonded IOBs	14	960	1%
Number of BUFG/BUFGCTRLs	1	32	3%

Table 5 indicates hardware resources utilized by FPGA for the case study including serial communication. Compared to a dedicated DIM, this consumes lesser hardware.

Table 6: Delay time of various FPGAs for both the case studies

Target Device	Device family	Delay time (ns)	Maximum clock (MHz)
Case study – Mixing of substances			
XC3S200	SPARTAN 3	9.558	104.62
XC6SLX4	SPARTAN 6	5.172	243.87
XC5VLX330T	VIRTEX 5	3.836	380.105
Case study – Traffic light control system			
XC3S200	SPARTAN 3	12.265	81.531
XC6SLX4	SPARTAN 6	4.729	211.475
XC5VLX330T	VIRTEX 5	4.222	236.855

Table 6 shows the delay time of various FPGAs to execute the logic for both the case studies. In traffic light control system, the design has used four simple on-delay timers and two off-delay timers. All these six timers will be functioning concurrently. PLC cannot activate more than one timer at a time. Thus, FPGA implementation helps to develop better quality product.

6.1 Comparison of Case study – PLC and FPGA

Proposed design has attempted to realize LD in terms of VerilogHDL code. Because, FPGA has been a better alternative for PLC. Hence, the proposed work has compared the performance of FPGA with PLC. GE Fenauc Series 90-30 35x Micro PLC has been considered for the purpose of performance comparison. The case study of substances mixing utilizes five inputs and three outputs. Scan time of five discrete inputs is 0.15*ms* as the scan time for one discrete input is 0.03*ms* [20]. Similarly, the scan time for three discrete outputs is 0.09*ms*. Hence the total scan time for five inputs and three outputs are 0.24*ms*. Delay time of XC3S200 FPGA identified from Table 6 is 9.558*ns* only. This time is inclusive of the scan time of input and output signals and the time required for the execution of logic.

This comparison has not included the time required for housekeeping, program execution, programmer service and the system communication. This increases the PLC sweep

time. Also, if the PLC is programmed for other applications in addition to the case study, then the PLC sweep time is increased further. This indicates that the PLC sweep time solely depends on the total number of rungs, inputs and outputs used in the LD and hence it is not constant. In the same way, FPGA can also be programmed for other applications in addition to the case study. In this case, the execution time remains constant as the hardware resources of FPGA for the case study and other applications are separate.

Apart from the time calculation, presented case studies can be viewed in the perspectives of quality of process operation. Case study of substances mixing involves mechanical parts whose response time will be in milliseconds. PLC may miss to identify and hence response to the changes, if it happens in between the sweep time of PLC. Whereas, FPGA will definitely identify the changes as its execution time is 9.558*ns*. However, if the changes happen in the time of less than 9.558*ns* then the FPGA also will miss to identify the changes. Thus the comparison proves that the FPGA is best suitable for high speed applications than that of the PLC.

Adding more number of digital inputs / outputs to the PLC increases its scan time. Digital input / output modules have in-built processors to handle these inputs / outputs. FPGA has both inputs and outputs and it provides the flexibility of treating input as output and vice-versa. PLC communicates to separate modules to receive digital inputs and send digital outputs. Table 5 shows that the device XC5VLX330T has 960 I/Os, whereas modules connected with PLC has the capacity of maximum of 64 inputs or outputs. Input and output modules are separate in PLC. Delay time by FPGA is in nano seconds and PLC scan time is in milli seconds [3, 21 – 22].

7. Conclusion

Case & event structure in LabVIEW, "assign" statement in VerilogHDL code are sufficient to create equivalent VerilogHDL code for the given LD. However, software package provided by PLC vendors to develop LD has numerous functional blocks. In VerilogHDL code, these functional blocks can be developed as library functions. Those library functions can be invoked as and when they are used in a LD. In this way, LD development using LabVIEW will become a complete solution for VerilogHDL code generation.

Data availability statement (DAS)

The datasets analyzed during the current study are available in the below references :

- Series 90^{TM}-30/20/Micro PLC CPU Instruction Set [online]. Available at: (http://support.ge-ip.com/support/resources/sites/GE_FANUC_SUPPORT/content/staging/DOCUMENT/0/DO704/en_US/1.0/gfk0467m.pdf) [June 2010].
- C. Henniga, K. Kneupner, D. Kinnab, Connecting programmable logic controllers (PLC) to control and data acquisition a comparison of the JET and Wendelstein 7- X approach. Fusion Engineering and Design. 87 (2012) 1972 – 1976.
- H. B. Mokadem, B. Bérard, V. Gourcuff, O. D. Smet, J. M. Roussel, Verification of a Timed Multitask System With UPPAAL. IEEE Trans. Autom. Sci. Eng. 7 (2010) 921 – 932.

The code generated during the current study is not publicly available but are available from the corresponding author on reasonable request.

Declarations

Funding

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Conflicts of interest/Competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this manuscript.

Availability of data and material (data transparency)

The datasets analyzed during the current study are available in the below references :

- Series 90^{TM}-30/20/Micro PLC CPU Instruction Set [online]. Available at: (http://support.ge-ip.com/support/resources/sites/GE_FANUC_SUPPORT/content/staging/DOCUMENT/0/DO704/en_US/1.0/gfk0467m.pdf) [June 2010].
- C. Henniga, K. Kneupner, D. Kinnab, Connecting programmable logic controllers (PLC) to control and data acquisition a comparison of the JET and Wendelstein 7- X approach. Fusion Engineering and Design. 87 (2012) 1972 – 1976.
- H. B. Mokadem, B. Bérard, V. Gourcuff, O. D. Smet, J. M. Roussel, Verification of a Timed Multitask System With UPPAAL. IEEE Trans. Autom. Sci. Eng. 7 (2010) 921 – 932.

Code availability (software application or custom code)

The code generated during the current study is not publicly available but are available from the corresponding author on reasonable request.

Authors' contributions

All authors contributed to the study conception and design, material preparation, data collection and analysis were performed by Dr. G. Dhanabalan and Dr. S. Tamil Selvi. The first draft of the manuscript was written by Dr. G. Dhanabalan and all authors commented on previous version of the manuscript. All authors read and approved the final manuscript.

References

[1] R. Bayindir, Y. Cetinceviz, A water pumping control system with a programmable logic controller (PLC) and industrial wireless modules for industrial plants—An experimental setup, ISA Trans. 50 (2011) 321 – 328.

[2] Y. Yan, H. Zhang, Compiling Ladder Diagram into Instruction List to comply with IEC 61131-3, Comput. Ind. 61 (2010) 448 – 462.

[3] R. Bayindir, Y. Cetinceviz, Design and implementation of an Internet based effective controlling and monitoring system with wireless field bus communications technologies for process automation—An experimental study. ISA Trans. 51 (2012) 461 – 470.

[4] A. Da'na S, Sagahyroon, A. Elrayes, A. R. Al-Ali, R. Al-Aydi, Development of a monitoring and control platform for PLC-based applications. Comput Stand. Interfaces. 30 (2008) 157 – 166.

[5] L. Morales-Velazquez, R. J. Romero-Troncoso, R. A. Osornio-Rios, G. Herrera-Ruiz, E. Cabal-Yepez, Open-architecture system based on a reconfigurable hardware–software multi-agent platform for CNC machines. J of Systems Archit. 56 (2010) 407 – 418.

[6] P. Zicari, P. Corsonello, S. Perri, G. Cocorullo, A matrix product accelerator for field programmable systems on chip. Microprocessors and Microsystems. 32 (2008) 53 – 67.

[7] A. Aiken, M. Fahndrich, Z. Su, Detecting races in Relay Ladder Logic programs. Int J STTT. 3 (2000) 93 – 105.

[8] G. Dhanabalan, S. Tamil Selvi, Design of parallel conversion multichannel analog to digital converter for scan time reduction of programmable logic controller using FPGA. Comput Stand. Interfaces. 39 (2015) 12 – 21.

[9] R. S. Moura, L. A. Guedes, Using basic State chart to program industrial controllers. Comput Stand. Interfaces. 34 (2012) 60 – 67.

[10] D. Gawali D, V.K. Sharma, FPGA based micro-PLC design approach. In: IEEE 2009 Advances in Computing, Control and Telecommunication Technologies Conference 2009, 28 – 29 December; Trivandrum, Kerala, INDIA: IEEE. pp. 660 – 663.

[11] J.I. Lee, S.W. Chun, S.J. Kang, Virtual prototyping of PLC-based embedded system using object model of target and behavior model by converting RLL-to-state chart directly. J. of Systems Archit. 48 (2002) 17–35.

[12] Celso F. Silva, Camilo Quintáns, Jose M. Lago, Enrique Mandado. An Integrated System for Logic Controller Implementation Using FPGAs. In: IEEE 2006 Industrial Electronics Conference; 6 – 10 November 2006; Paris, FRANCE: IEEE. pp. 195 – 200.

[13] W. T. Sung, Y. C. Hsu, Designing an industrial real-time measurement and monitoring system based on embedded system and ZigBee. Expert Systems with Applications. 38 (2011) 4522 – 4529.

[14] P. Martín, E. Bueno, J. Rodríguez, O. Machado, B. Vuksanovic, An FPGA-based approach to the automatic generation of VHDL code for industrial control systems applications: A case study of MSOGIs implementation, Mathematics and Computers in Simulation 91 (2013) 178–192.

[15] A. Milik, On ladder diagram compilation and synthesis of FPGA implemented reconfigurable logic controller. Advances in Electrical and Electronic Engineering. 12 (2014) 443 – 451.

[16] John, Karl-Heinz, Tiegelkamp, Michael, IEC 61131 – 3: Programming Industrial automation systems, Springer-Verlag, 2010.

[17] D. Rzonca, J. Sadolewski, B. Trybus, Prototype environment for controller programming in the IEC 61131-3 ST Language. Computer science and information systems, 4 (2007) 133 - 148.

[18] S. Yu, X. Xu, An input and output monitoring system for FPGA based hardware PLC. International journal of Engineering and Industries, 3 (2012) 34 – 44.

[19] D. Patel, J. Bhatt, S. Trivedi, Programmable logic controller performance enhancement by field programmable gate array based design. ISA Trans, 54 (2015) 156 – 168.

[20] Series 90™-30/20/Micro PLC CPU Instruction Set [online]. Available at: (http://support.ge-ip.com/support/resources/sites/GE_FANUC_SUPPORT/content/staging/DOCUMENT/0/DO704/en_US/1.0/gfk0467m.pdf) [June 2010].

[21] C. Henniga, K. Kneupner, D. Kinnab, Connecting programmable logic controllers (PLC) to control and data acquisition a comparison of the JET and Wendelstein 7- X approach. Fusion Engineering and Design. 87 (2012) 1972 – 1976.

[22] H. B. Mokadem, B. Bérard, V. Gourcuff, O. D. Smet, J. M. Roussel, Verification of a Timed Multitask System With UPPAAL. IEEE Trans. Autom. Sci. Eng. 7 (2010) 921 – 932.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- tlcvideo.mp4