OpenPLC: An Open Source Alternative to Automation

Thiago Rodrigues Alves, Mario Buratto, Flavio Mauricio de Souza, Thelma Virginia Rodrigues
Departamento de Engenharia Eletrônica e de Telecomunicações
PUC Minas
Belo Horizonte, Brazil
thiagoralves@gmail.com

Abstract

Companies are always looking for ways to increase production. The elevated consumerism pushes factories to produce more in less time. Industry automation came as the solution to increase quality, production and decrease costs. Since the early 70s, PLC (Programmable Logic Controller) has dominated industrial automation by replacing the relay logic circuits. However, due to its high costs, there are many places in the world where automation is still inaccessible. This paper describes the creation of a low-cost open source PLC, comparable to those already used in industry automation, with a modular and simplified architecture and expansion capabilities. Our goal with this project is to create the first fully functional standardized open source PLC. We believe that, with enough help from the open source community, it will become a low cost solution to speed up development and industrial production in less developed countries.

Keywords—PLC; OpenPLC; Automation; MODBUS; Open source

I. Introduction

In early 60s, industrial automation was usually composed of electromechanical parts like relays, cam timers and drum sequencers. They were interconnected in electrical circuits to perform the logical control of a machine. To change a machine logic was to make an intervention on its electrical circuit, which was a long and complicated process.

In 1968, the Hydra-Matic of General Motors requested proposals for an electronic replacement for hard-wired relay systems. The winning proposal came from Bedford Associates with their 084 project. The 084 was a digital controller made to be tolerant to plant floor conditions, and was latter known as a Programmable Logic Controller, or simply PLC [1].

Within a few years, the PLC started to spread all over the automotive industry, replacing relay logic machines as an easier and cheaper solution, and becoming a standard for industrial automation.

There is a strict relation between automation and development. In less developed countries, the greatest barriers are knowledge and cost. Industrial controllers are still very expensive. Companies don't provide detailed information about how these controllers work internally as they are all closed source.

The OpenPLC was created to break these two barriers, as it is fully open source and open hardware. It means that anyone can have access to all project files and information for free. This kind of project helps spread technology and knowledge to places that need the most. Also, the OpenPLC is made with inexpensive components to lower its costs, opening doors to automation where it wasn't ever possible before.

II. The PLC Architecture

The PLC, being a digital controller, shares common terms with typical PCs, like CPU, memory, bus and expansion. But there are two aspects of the PLC that differentiate them from standard computers. The first one is that its hardware must be sturdy enough to survive a rugged industrial atmosphere. The second is that its software must be real time.

A. Hardware

With the exception of "Brick PLCs" that are not modular, the hardware of a usual PLC can be divided into five basic components:

- Rack
- Power Supply
- CPU [Central Processing Unit]
- Inputs
- Outputs

Like a human spine, the rack has a backplane at the rear allowing communication between every PLC module. The power supply plugs into the rack providing a regulated DC power to the system.

The CPU is probably the most important module of a PLC. It is responsible for processing the information received from input modules and, according to the programmed logic, send impulses to the output modules. The CPU holds its program on a permanent storage, and uses volatile memory to perform operations. The logic stored in CPU's memory is continuously processed in an infinite loop. The time needed to complete a cycle of the infinite loop is called scan time. A faster CPU can achieve shorter scan time.

Input modules are used to read signals of sensors installed at the field. There are many types of input modules, depending on the sensor to be read, but they can generally be split into two categories: analog and digital.

Digital input modules can handle discrete signals, generated by devices that are either on or off. Analog input modules convert a physical quantity to a digital number that can be processed by the CPU. This process of conversion is usually made by an ADC [Analog to Digital Converter] inside the analog input module. The type of the physical quantity to be read determines the type of the analog input module. For example, depending on the sensor, the physical value can be expressed in voltage, current, resistance or capacitance.

Similarly to the input modules, output modules can control devices installed at the field. Digital output modules

978-1-4799-7193-0/14/$31.00 ©2014 IEEE

can control devices as if on-off switches. Analog output modules can send different values of voltage or current to control position, power, pressure or any other physical parameter.

As the most significant feature of a PLC is robustness, each module must be designed with protections such as short circuit, over current and over voltage protections. It is also important to include filter against RF noise.

B. Software

PCs, by design, are made to handle different tasks at the same time. However, they have difficulty handling real time events. To have an effective control, PLCs must be real time. A good definition of real time is "any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period" [2]. Real time systems don't necessarily mean to be fast. They just need to give an answer before the specified period known as deadline. Systems without real time facilities cannot guarantee a response within any timeframe. The deadline of a PLC is its scan time, so that all responses must be given before or at the moment scan reaches the end of the loop.

There are many accepted languages to program a PLC, but the most widely used is called ladder logic, which follows the IEC 61131-3 standard [3]. Ladder logic (see Fig. 1) was originally created to document the design and construction of relay logic circuits. The name came from the observation that these diagrams resemble ladders, with two vertical bars representing rails and many horizontal rungs between them. These electrical schematics evolved into a programming language right after the creation of the PLC, allowing technicians and electrical engineers to develop software without additional training to learn a computer language, such as C, BASIC or FORTRAN.

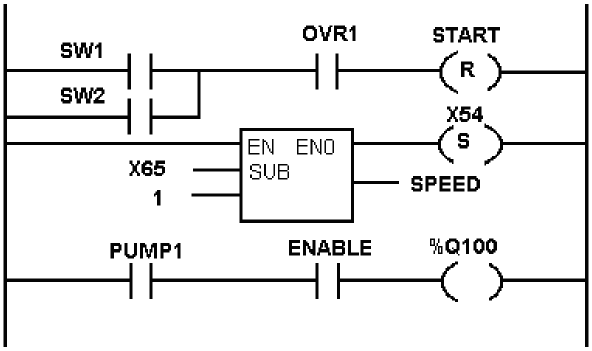


Fig. 1. Example of a ladder logic diagram

Every rung in the ladder logic represents a rule to the program. When implemented with relays and other electromechanical devices, all the rules execute simultaneously. However, when the diagram is implemented in software using a PLC, every rung is processed sequentially in a continuous loop (scan). The scan is composed of three phases: 1) reading inputs, 2) processing ladder rungs, 3) activating outputs. To achieve the effect of simultaneous and immediate execution, outputs are all toggled at the same time at the end of the scan cycle.

III. The OpenPLC Hardware Architecture

The OpenPLC (see Fig. 2) was created based on the architecture of actual PLCs on the market. It is a modular system, with expansion capabilities, an RS-485 bus for communication between modules and hardware protections.

To create the first OpenPLC prototype, four boards were built:

- Bus Board
- CPU Card
- Input Card
- Output Card

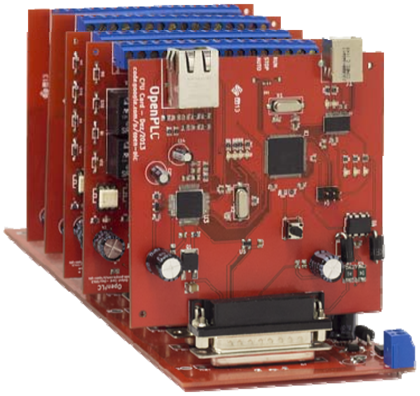


Fig. 2. The OpenPLC Prototype

A. Bus Board

The bus board acts like a rack, with an integrated 5VDC power supply. Each module connects to the bus board through a DB-25 connector. The communication between modules is made over an RS-485 bus, whose lines are on the bus board. Caution was taken, while routing the RS-485 lines, to avoid communication problems. Fig. 3 shows the pins and connections of each slot of the bus board. The 24V and RS-485 ground was separated from the rest of the circuit ground to isolate short circuits on these lines.

To allow more current to flow through the power lines, the respective pins were duplicated. Three pins were used for physical address, so that the module connected on a particular slot would know its physical position on the bus board. These pins were called D0, D1 and D2, being hardcoded with logic 1 or 0 in a binary sequence, creating different numbers from 0 to 7, one number for each slot.

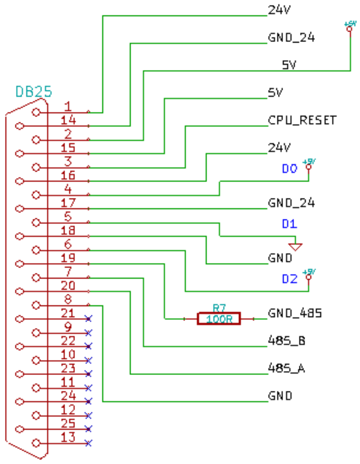


Fig. 3. Bus Board DB-25 connections

B. *CPU Card*

The OpenPLC's brain is the CPU card. It was important to use a processor that was inexpensive, fast enough to handle all PLC operations, and most importantly, actively supported by the open source community. After some research, the processor selected was the AVR ATmega2560. This microcontroller is a "high-performance, low-power Atmel 8-bit AVR RISC-based microcontroller that combines 256KB ISP flash memory, 8KB SRAM, 4KB EEPROM, 86 general purpose I/O lines, 32 general purpose working registers, real time counter, six flexible timer/counters with compare modes, PWM, 4 USARTs, byte oriented 2-wire serial interface, 16-channel 10-bit A/D converter, and a JTAG interface for on-chip debugging. The device achieves a throughput of 16 MIPS at 16 MHz and operates between 4.5-5.5 volts" [4].

The biggest reason for this choice was that the ATmega2560 is used on the Arduino family [5], a large open source community for rapid electronic prototyping, with an advanced programming language called Wiring. By using this processor we made the OpenPLC compatible with Arduino code, including hundreds of libraries written for it.

The CPU card also includes another important IC (Integrated Circuit), the Wiznet W5100, responsible for Ethernet communication. The Wiznet W5100 supports hardwired TCP/IP Protocols like TCP, UDP, ICMP, IPv4 ARP, IGMP, PPPoE and Ethernet 10BaseT/100BaseTX, has 16KB of internal memory for Tx/Rx buffers and accepts serial (over SPI) or parallel interface. This is also the Arduino Ethernet Shield official IC, enabling us to reuse all the code written for it on the OpenPLC.

In order to communicate with the PC and download programs, the OpenPLC uses an USB port. The FT232RL from FTDI Devices converts Serial Rx/Tx lines to USB standard. The Arduino Mega bootloader is used to upload code to the CPU over the USB circuit.

C. *Input Card*

The Input card is a digital input module for the OpenPLC. To process the digital inputs read by the conditioning signal circuit and send them to the CPU card, the input card uses the AVR ATmega328P, a microcontroller with the same core of the CPU card. This made the reutilization of parts of code written for the CPU card, especially code related to communication over the RS-485 bus possible.

The input signal conditioning circuit is composed mainly by an optocoupler, used to isolate the input signals and the control signals. The circuit of each input can be seen on Fig. 4.

When a stimulus is made between E1+ and E1-, a current flow through the input resistor and activates the internal LED of the optocoupler. The photons emitted by the internal LED are sensed by the phototransistor, which creates a path for the current from 5VCD to ground, sending logic 0 to inverter's input. As the inverter must invert the logic signal, a logic 1 is received by the microcontroller, indicating that a digital stimulus was made at the input.

The input card has 8 isolated input circuits, so that each module can read up to 8 digital signals at the same time. The state of each input is sent to the CPU card, over the RS-485 bus, to be processed according to the ladder logic.

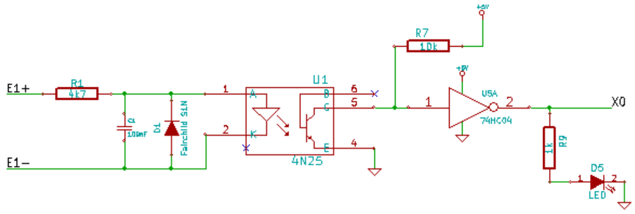


Fig. 4. Isolated input circuit from the Input Card

D. *Output Card*

Each Output card has 8 relay-based outputs driving up to 8 loads at the same time. It has double isolated outputs, as they are isolated by an optocoupler (just like the Input card) and the relay itself, which gives an additional layer of isolation. Fig. 5 shows the circuit of one isolated output from the Output card.

As digital processors are better sinking current than sourcing, the cathode of the optocoupler's internal LED is connected to an output pin on the ATmega328P. While the output pin remains with logic 1, no current flows through the LED. If the output pin goes to logic 0, a current is drawn on that pin, activating the optocoupler's internal LED. The internal phototransistor is connected to an external BC817 transistor in a Darlington configuration to increase gain. When photons are sensed by the internal phototransistor, both transistors are polarized, energizing the relay's coil. Without photons, there isn't any current flowing through the coil, and the relay remains off.

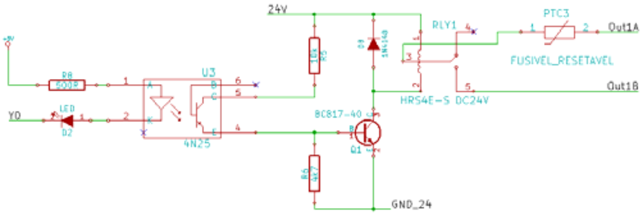


Fig. 5. Isolated output circuit from the Output Card

E. Protections

There are five types of protections used in the OpenPLC circuit:

- Current limiting protection with PPTC [Polymeric Positive Temperature Coefficient]
- Over-voltage protection with TVS [Transient Voltage Suppression diode]
- Ground isolation
- Reverse polarity protection
- Noise filters

Every input and output (including the power input at the Bus board) has protection against over-voltage and short circuit. These protections are achieved by using a PPTC in series with the circuit input and a TVS diode in parallel.

When a high current flows through the PPTC, it reaches a high resistance with a low holding current, protecting the circuit in series. When the current is removed, it cycles back to a conductive state, enabling the circuit to work properly.

Optocouplers and relays were used to isolate high power circuits from control logic. The filled zones were connected to ground and only the low power zones of the board were filled. To isolate the communication and 24V grounds from the filled zone, zero ohm resistors were used.

To protect against reverse polarity on inputs, diodes were connected in series to allow current flow in only one direction. Also, capacitors were used in parallel to ground to filter noise from sensitive devices.

IV. The OpenPLC Software Architecture

What differentiates a PLC from any other robust digital controller is its ability to be programmed in some standardized languages. According to [3], the IEC 61131-3 standard defines five languages on which PLCs can be programmed:

- FBD [Function Block Diagram]
- Ladder Diagram
- Structured Text
- Instruction List
- SFC [Sequential Function Chart]

The most widely used language in PLC is the Ladder Diagram. PLCs from different manufacturers might not have all the five programming languages available, but they certainly have the Ladder Diagram as one of the options.

For this reason, it was important to develop a software that was able to compile a ladder diagram into a code that could be understood by the CPU of the OpenPLC. The solution was partially based on LDmicro [6], an open source ladder editor, simulator and compiler for 8-bit microcontrollers. It generates native code for Atmel AVR and Microchip PIC16 CPUs from a ladder diagram.

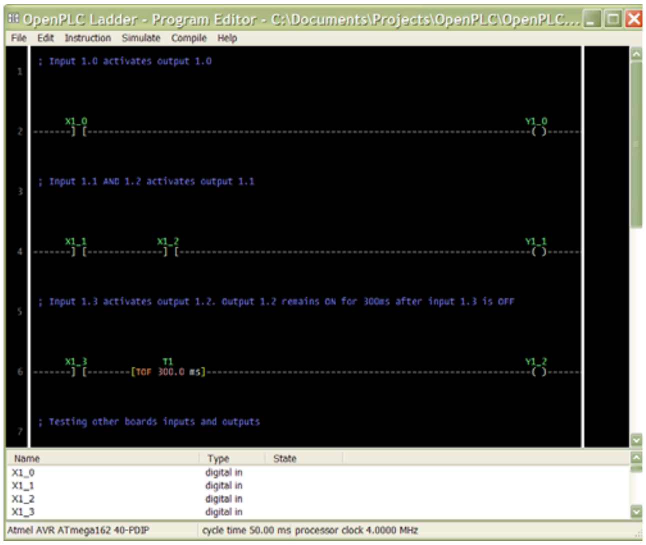
Unfortunately, the OpenPLC CPU uses the ATmega2560, which is not supported by the original LDmicro software. Also, the generated code contains only the ladder logic converted to assembly instructions. The OpenPLC has many other functions to perform, such as communication over Ethernet for MODBUS-TCP supervisory systems, RS-485 and USB, individual modules control, error messages generation and so on.

For this reason, it was necessary to create an intermediate step before the final compilation in which the ladder diagram had to be combined with the OpenPLC firmware. Doing so, the final program would contain both the ladder logic and the OpenPLC functions. One of the outputs generated by the LDmicro for the Ladder Diagram was ANSI C code. So, instead of having machine code for a specific processor, an ANSI C code that could be compiled for any platform was generated. The only thing that had to be provided using this method was a C header to link the generated ANSI C functions and the target system.

The OpenPLC Ladder Editor (Fig. 6) was created to fulfill these tasks. Basically, the OpenPLC Ladder Editor is a modified version of the LDmicro, with reduced instructions (processor-specific instructions had to be removed), no support to direct compiling (it only generates ANSI C code) and a tool that can automatically link the generated ANSI C code with the OpenPLC firmware, compile everything using AVR GCC and upload the compiled software to the OpenPLC.

The compiler tool is called every time the "compile" button is clicked. While the code for the LDmicro was created using C++, the compilation tool was created using C# .net, a very robust and modern language.

The final result is a binary program uploaded to the OpenPLC CPU, containing both the ladder logic and the functions of the OpenPLC firmware.


OpenPLC Ladder - Program Editor - C:\Documents\Projects\OpenPLC\OpenPLC...
File
Edit
Instruction
Simulate
Compile
Help

```
; Input 1.0 activates output 1.0

       X1_0                                                              Y1_0
-------] [---------------------------------------------------------------( )------

; Input 1.1 AND 1.2 activates output 1.1

       X1_1          X1_2                                                Y1_1
-------] [-----------] [-------------------------------------------------( )------

; Input 1.3 activates output 1.2. Output 1.2 remains ON for 300ms after input 1.3 is OFF

       X1_3          T1                                                  Y1_2
-------] [--------[TOF 300.0 ms]-----------------------------------------( )------

; Testing other boards inputs and outputs
```

Name	Type	State
X1_0	digital in	
X1_1	digital in	
X1_2	digital in	
X1_3	digital in	

Atmel AVR ATmega162 40-PDIP | cycle time 50.00 ms processor clock 4.0000 MHz

Fig. 6. OpenPLC Ladder Editor software running on a PC

A. MODBUS Communication

MODBUS is an industry standard protocol for automation devices. Although, the message format is maintained, there are some variations of this protocol depending on the physical interface it will be used on. As the OpenPLC has Ethernet over TCP-IP, it was implemented support for the MODBUS-TCP protocol. Only the most used functions of the protocol were implemented, as shown next:

- FC01 - Read Coil Status
- FC02 - Read Input Status
- FC03 - Read Holding Registers
- FC05 - Force Single Coil
- FC15 - Force Multiple Coils

B. *Boards Communication*

To become a modular system, each module of the OpenPLC must have a way to communicate with the CPU. The RS-485 bus is the physical protocol through which messages are sent. But it was necessary to create a protocol on the application layer, to standardize the messages sent and received.

The protocol created was called OPLC Protocol. It is a simple protocol that encapsulates each message sent or received with information about destination, size of the message and function to be executed.

TABLE I. OPLC PROTOCOL HEADER

Start	Size	Function	Address	Data
1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

Every message starts with a start byte, which is always 0x7E. The receiver will only process the message after receiving the start byte. The size field must contain the size (in bytes) of the Data field only. The function field is related to the data field. It means that what the receiver will do with the data received depends on the function. Five functions were implemented for the OPLC Protocol:

- 0x01 – Ask for the card type
- 0x02 – Change card logical address
- 0x03 – Read discrete inputs
- 0x04 – Set discrete outputs
- 0x05 – error message

The address field may have the logic or the physical address of the card, according to the function requested. For example, the functions 0x01 and 0x02 are addressed to the physical address, because they are related to low level commands, such as get card information or change the logical address.

V. RESULTS

To evaluate the OpenPLC as a real PLC, a benchmark had to be made comparing it with another controller. This was achieved using a model of a five floor building with an elevator originally controlled by a Siemens S7-200 PLC.

Modifications were made to the model enabling it to interchange PLCs easily for the tests. The elevator is moved by a DC motor attached to it. There are limit switches on every floor to indicate elevator's position. Also, limit switches were installed at the top and bottom of the building to prevent the elevator to move over the permitted range. Lights indicators on every floor were used to visually indicate when the elevator stops at the respective floor. Five push buttons were used to call the elevator to the desired floor.

The ladder diagram for this task was already written for the Siemens PLC using the Siemens Step 7 platform. It used 13 digital inputs and 10 digital outputs to fully control the model. The diagram was printed and the exactly same diagram was written for the OpenPLC using the same logic blocks, see Figure 10. The OpenPLC Ladder Editor was used to compile, simulate and upload the diagram to the OpenPLC.

During tests, a bug on the ladder diagram was found. If the user held the push button related to the floor on which the elevator was located while pushing another button to send it to another floor, the system hung with an infinite loop. As expected, the OpenPLC behaved exactly the same way as the Siemens PLC, presenting the same bug. After correcting the ladder on both controllers, each one operated flawlessly. The response of diverse stimulus on each PLC was identical on every tested situation.

VI. CONCLUSION

The open source community is growing stronger every day. There are many projects, from software to hardware with contributions from people all around the world. Creating an open source industrial controller from scratch is a very bold task. But thanks to the support of the open source community like the *Arduino* and *LDmicro* it was possible to create a prototype of a functional PLC comparable with a standardized industry controller. During tests, the OpenPLC behaved exactly the same way as other controllers, given the same input impulses. The MODBUS-TCP communication was tested using SCADA software from different vendors. It was possible to read inputs and outputs and force outputs as it would be on any other PLC.

Our next big step is to use our OpenPLC in a field application, evaluating its robustness, versatility and ease of use for the user.

REFERENCES

[1] P.E. Moody and R.E. Morley, “How Manufacturing Will Work in the Year 2020”, Simon and Schuster.

[2] R. Oshana and M. Kraeling, “Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications”, 1st ed. Newnes, 2013 pp.12-20.

[3] K.H. John and M. Tiegelkamp, “IEC 61131-3: Programming Industrial Automation Systems,” 2nd ed. Springer, 2010 pp.147-168.

[4] Atmel Corporation, “ATmega2560,” Atmel.com. 2014. 8 Jul. 2014. http://www.atmel.com/devices/atmega2560.aspx.

[5] Arduino, “Arduino MEGA ADK,” arduino.cc. 2014. 8 Jul. 2014. http://arduino.cc/en/Main/ArduinoBoardMegaADK.

[6] J. Westhues, “Ladder Logic for PIC and AVR,” cq.cx. 2014. 8 Jul. 2014. http://cq.cx/ladder.pl.