eLABS PROJECT

Traffic Signals for Model Railroads

Programming with ladder diagrams

By **Rob van Hest** (The Netherlands)

It all started with a simple question: could the author make a controller for traffic signals on a model railroad? That led to the rediscovery of ladder diagrams as a simple and effective way to create program logic.

PROJECT INFO

- LED traffic signal
- Microcontroller
- Ladder diagram
- ➡ entry level
- intermediate level
- expert level
- 1 hour approx.
- Normal soldering tools, programmer (optional)
- €25 / £20 / $30 approx.

Nowadays when you need to automate something, the solution is obvious: you take an Arduino or Raspberry Pi module or choose one of the countless other single-board computers, put together a shield, a hat or whatever you want to call it, write some code, and you're done.

But using a complete minicomputer just to control a traffic signal is naturally a bit like using a cannon to swat flies. There must be an easier way, with a microcontroller and a little bit of I/O.

Features

- Traffic signal for model railroads
- Ladder diagram programming
- Free programming environment
- Suitable for 12 V and 24 V
- All components through-hole

PLC

In the real world, traffic signals are often operated by programmable logic controllers (PLCs), which are also used extensively for industrial control tasks. PLCs are usually programmed using ladder diagrams – and for good reason.

Microcontrollers are normally programmed using a high-level language such as C, which should not present any difficulties for most Elektor readers. However, service technicians and electricians in the industrial sector are not as fluent in these programming languages. On the other hand, they are perfectly at home with relay-based control circuits. That is the main advantage of ladder diagrams: if you can build a

circuit using electromechanical relays, you can program a PLC with the aid of ladder diagrams.
The author actually worked with ladder diagrams many years ago, and with a very convenient ladder compiler that runs perfectly under Windows 7 now available, the choice was easy: build the traffic signal controller as a mini-PLC and program it with ladder diagrams.

An introduction to ladder diagrams

As the name suggests, a ladder diagram looks something like a ladder with several rungs. Each rung consists of one or more switch contacts, which are drawn from left to right, and an actuator (the coil of a relay). For this brief introduction, we remain within the realm of electromagnetic relays. The switch contacts are depicted as follows:

```
---] [---
```
Normally open (NO) contact

```
---]\[---
```
Normally closed (NC) contact

As you can see, switch contacts are represented by square brackets in "reverse" order.

Actuators are indicated by round brackets (parentheses):

```
---( )---
```
Normally inactive actuator (unenergized relay coil)

```
---(/)---
```
Normally active actuator (energized relay coil)

As you probably already guessed, we can use these symbols to depict logical relationships.

A logic AND looks like this:

```
---] [----------------] [---------------( )---
   sw1                 sw2               motor
```

Here the motor is only activated (switched on) when contact sw1 and contact sw2 are both closed.

A logic OR takes the following form:

```
---+---] [---+---------------------------( )---
   |   sw1   |                           motor
   |         |
   +---] [---+
       sw2
```

A logic NOT (inversion) is indicated by a slash:

```
---] [----------------]/[----------------( )---
   sw1                 sw2                motor
```

Here the motor is activated when contact sw1 is closed and contact sw2 is not closed.
Using this method, you can construct a logic controller step by step (rung by rung) using switches and relays.

A mental exercise

As a sort of mental exercise, let's see if we can use this knowledge to do something useful – without actually drawing a circuit diagram.
Anyone interested in electronics is probably familiar with the "quizmaster" circuit: several contestants hear the quizmaster ask a question, and the one who presses the button first gets the chance to answer the question and score a point.
Now let's try to implement this circuit in the form of a ladder diagram.
We start with a master signal called "enable," which is only active when none of the contestants has pressed their button. When a contestant presses the button, an associated bistable relay is set and a lamp or something similar is switched on. The corresponding ladder rung looks like this:

```
  Yenable      XAplay              YAplay
----] [----+----] [-----------------(S)----
           |
           |   XBplay              YBplay
           +----] [-----------------(S)----
           |
           |   XCplay              YCplay
           +----] [-----------------(S)----
```

The "Yenable" signal is only true (active) when none of the contestant buttons is pressed:

```
   YAplay     YBplay     YCplay   Yenable
----]/[--------]/[--------]/[-------( )----
```

Of course, we also need a way to reset everything for the next round. That could look something like this:

```
  Yenable     Xreset               YAplay
----]/[---------] [----+------------(R)----
                       |
                       |           YBplay
                       +------------(R)----
                       |
                       |           YCplay
                       +------------(R)----
```

Finally, "END" indicates that we are done:

```
----[END]----------------------------------
```

As you can see, it is easy to construct a logic circuit with just a bit of logical thinking, without putting a single mark on a piece of paper.

LDmicro

That's all well and good, but we naturally need a programming environment if we want to draw ladder diagrams and compile them to produce hex files that can be loaded into a microcontroller. That is exactly what we want to present here. The program in question is called LDmicro, and it

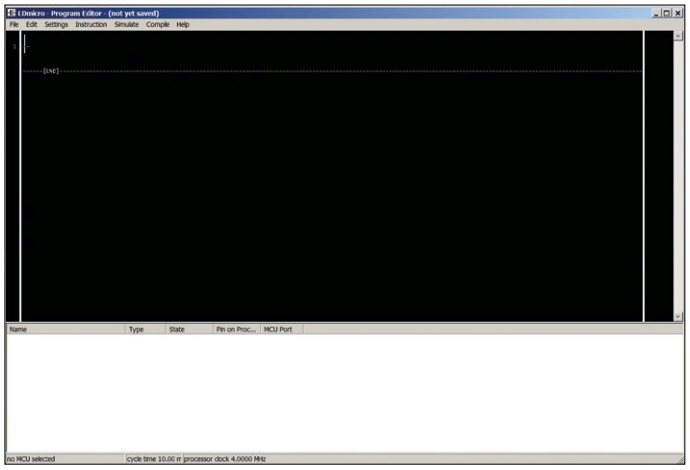


Figure 1. LDmicro starts with a clean sheet.

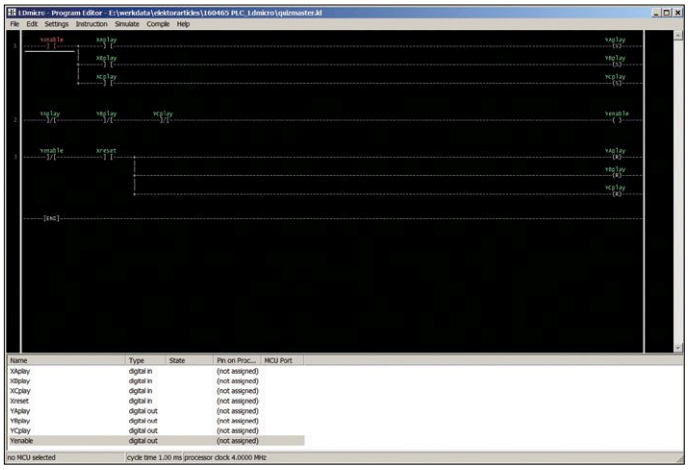


Figure 2. The quizmaster mental exercise in LDmicro.

works well, looks good, and does not cost anything [1]. The program takes the form of an executable file that does not have to be separately installed. **Figure 1** gives an impression of the user interface.
The diagram for our quizmaster experiment is shown in **Figure 2**. As you can see, each rung of the ladder is assigned a number, and the various contacts and actuators are listed in the bottom pane as inputs and outputs (this is done fully automatically).

The program documentation is exemplary in all regards, so it will not take you very long to learn to use the program and there is no need to go into the details of how to enter the various components of the diagram.
In order to translate the ladder diagram into a file that can be used to program a microcontroller (in other words, to compile it), you first have to select the target microcontroller as illustrated in **Figure 3**. As you can see, the program supports a variety of microcontrollers. Here the author selected the PIC16F628A, simply because he had a large number of these devices on hand.

Finally, before compiling the program you have to assign the various signals to the microcontroller pins. To do so, click on an input or output in the bottom pane and select the appropriate pin in the resulting pop-up window, as illustrated in **Figure 4**. Then compile the program and download the resulting hex file to the microcontroller. The author used a Velleman K8048 programmer for this, but the TL866A universal programmer available in the Elektor Store [3] also works well.
The download for this article [2] also includes the quizmaster example in a more elaborate version with an output for a buzzer. We leave it up to you (as a good exercise) to figure out how that works.

Getting down to business

Let's get back to where we started with this article: a traffic signal for a model railway. The object here is to safeguard an intersection, so we need four traffic signals in total, operating in pairs.
This could of course be constructed with a ring counter (using a couple of CD4017 ICs, for example) and a diode

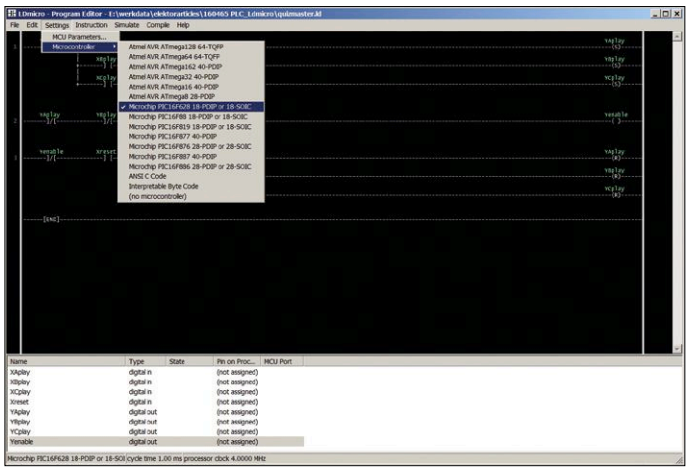


Figure 3. You can select the right microcontroller under Settings. LDmicro supports many different types.

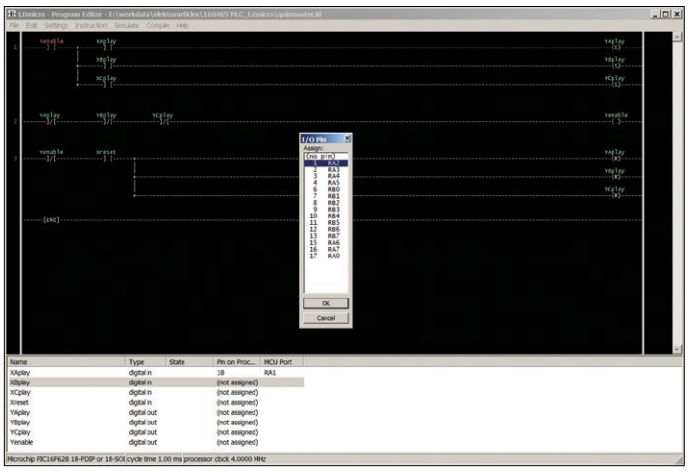


Figure 4. Assigning inputs and outputs is very straightforward.

matrix, but that has two disadvantages: you need a relatively large number of components, and it is difficult to modify the traffic signal behavior. We will therefore build a simple PLC that can be programmed using ladder diagrams.
The core of the program actually consists of a counter which counts 30 steps (from 0 to 29), of which 15 are allocated to one set of signals (which we call north/south) and the other 15 to the other set (east/west).
This counter can be programmed in LDmicro as a single rung.

```
  Rclock        Ton              Toff             Rclock
---]/[----[TOF 500.0 ms]--[TON 500.0 ms]--+-------( )------
                                           |
                                           |     Ccycle1
                                           +---{CTC 0:29}---
```

Here we employ two delays of 500 ms each, resulting in a cycle time of 1 second (1 Hz repeat rate). To keep track of the cycles, we use a counter which counts from 0 to 29 and then resets to 0.
The timing of the signal lights is determined by the counter state. For example, for the north/south pair we have:

1 or less:	red
2–8:	green
9:	yellow
10 or more:	red

This is fairly easy to translate into a ladder diagram:

```
   [Ccycle1 >=]     [Ccycle <]        Ygreen
---[ 2        ]-----[ 9      ]----------( )---

   [Ccycle1 ==]                       Yyellow
---[ 9        ]-------------------------( )---

   [Ccycle1 >=]                        Yred
---[ 10       ]--+----------------------( )---
                 |
   [Ccycle1 <]   |
---[ 2       ]---+
```

For the east/west pair, we simply add 15 to the previous counter states:

16 or less:	red
17–23:	green
24:	yellow
25 or more:	red

This is the main part of the traffic signal program. The complete program is included in the download file. There we added a pedestrian signal, an option for selecting either the signal sequence with yellow before green as well as yellow before red (as used in Germany) or the sequence without yellow before green, and a night mode (all signals blinking yellow). **Figure 5** shows part of the complete program in LDmicro, as well as the pin assignments.

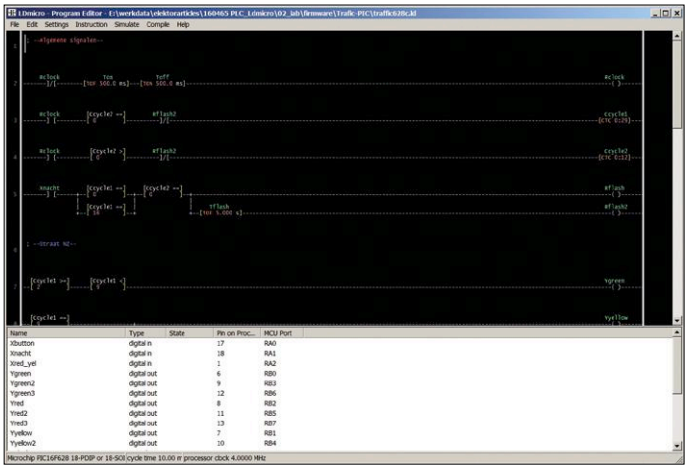


Figure 5. The traffic signal program in LDmicro.

Pocket-size traffic signals

The hardware

Figure 6 shows the schematic diagram of the traffic signal controller. This does not need much explanation. The supply voltage input is on connector K7. The supply voltage stated on the schematic is 12 V_{DC}, which is commonly used for model railroads, but the circuit can be used unchanged with supply voltages from 9 V to 24 V. In the latter case, it is a good idea to fit a small finned heat sink on voltage regulator IC2 (type 78L05). The 5 V output from IC2 is used solely to power the microcontroller. Diode D1 provides reverse-polarity protection.

The circuit is built around the microcontroller IC1. No crystal is used here; the clock signal for the processor is generated by the free-running internal oscillator of IC1. It is more that sufficiently accurate for our purposes.
The inputs (connectors K4–K6) are protected by voltage dividers and series resistors, so that voltages up to a maximum of 24 V can be used as input signals without any problem.

The microcontroller outputs are buffered by the driver IC3 (type ULN2803). It should be familiar to most readers. The outputs of this IC switch to ground. The driver can handle approximately 60 mA per output when all outputs are active at the same time. That is more than enough for model railroad traffic signals. If you need more current, you can connect relays to the outputs, and the free-wheeling diodes necessary for this are already integrated in the IC.
Jumper JP1 selects between normal operating mode (2-3) and programming mode (1-2). In programming mode, the microcontroller can be programmed through connector K8 without removing it for the circuit (in-system programming, ISP). Of course, you can always program the microcontroller in a separate programmer.

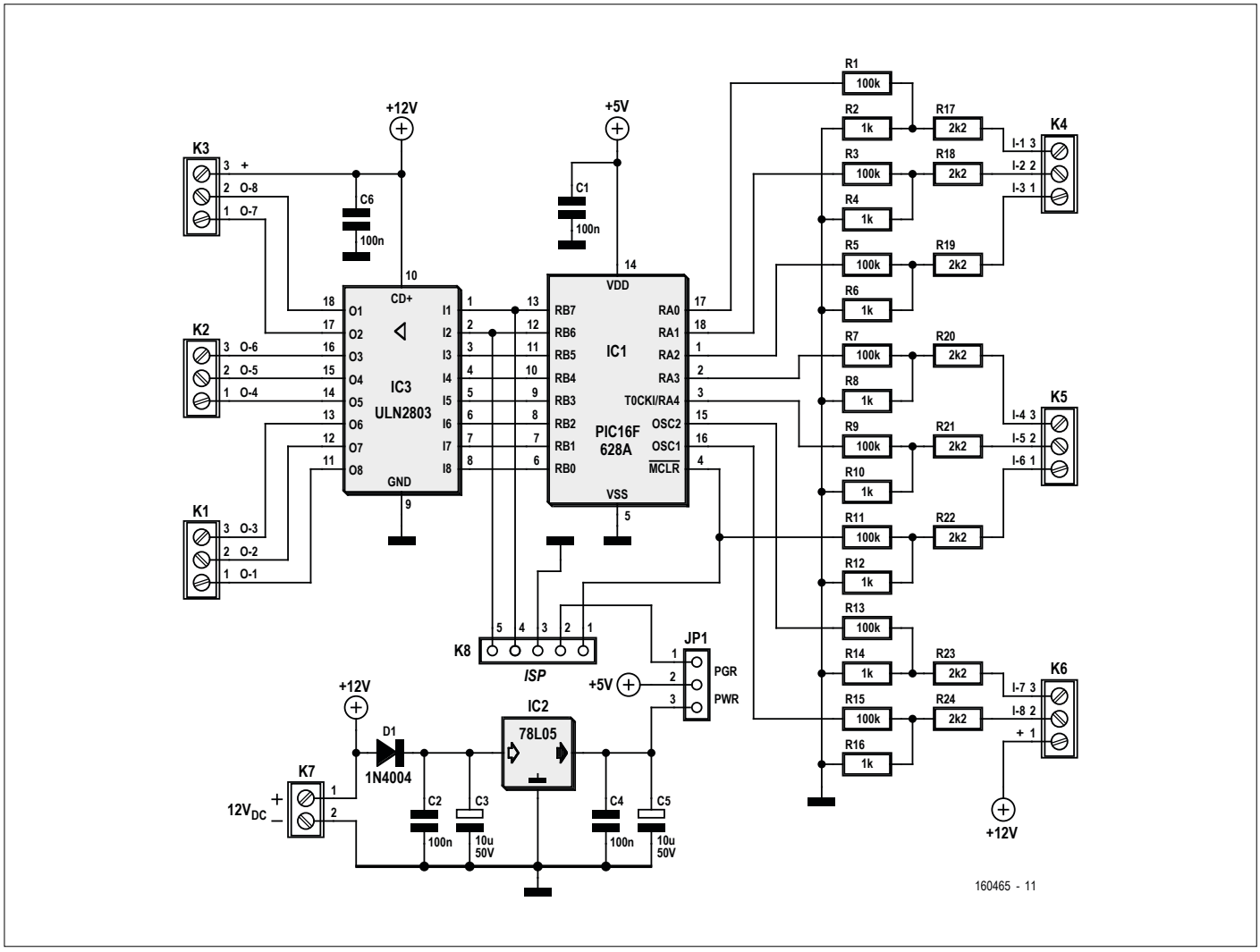


Figure 6. Schematic diagram of the traffic signal controller. This is actually a small general-purpose PLC, which with suitable software can control traffic signals on a model railroad.

Construction and use

For the traffic signal controller we designed a PCB (single-sided) using exclusively conventional through-hole components (**Figure 7**). Assembling the board should not present any problems for most readers. Start with the sockets for IC1 and IC3, then mount the low-profile components (diodes and resistors), followed by the capacitors and connectors. After a careful visual inspection, you can connect a 12 V power source to connector K7 (pay attention to the right polarity) and check for a clean and stable 5 V supply voltage on pin 3 of JP1. If that test is passed, switch off the power and place a jumper on pins 2 and 3 of JP1 (where it can remain forever if you never need to program the microcontroller in system) and insert the microcontroller (IC1) and the driver (IC3) in their sockets. Next, connect the LEDs of the various traffic signals and the switches as indicated in **Figure 8**. Note that if you want to use a 24 V supply voltage instead of 12 V, it is probably a good idea to increase the series resistor values for the LEDs to 2.2 kΩ. Using the controller is easy: press S1 briefly to activate the pedestrian light (just like real life, where you always have to press a button before you can cross safely).

Switch S2 activates night mode; when it is switched off, the controller sets all lights to red for safety before changing to normal mode. Finally, S3 allows you to select either "NA/UK" or "Germany" signal sequence.

One final tip: If you use a 24 V supply voltage, it is a good idea to increase the value of R17–R24 to 4.7 kΩ. This will prevent the voltage on pin 4 of the microcontroller from rising too high. Although that would not cause any serious damage, it might unintentionally put the microcontroller in programming mode.

Conclusion

A model railroad traffic signal is of course a perfectly legitimate application for this circuit. However, we hope this project has aroused your interest in programming with ladder diagrams — using the board described here (with its wealth of inputs and outputs), you can implement a wide range of attractive applications without any programming language experience. Be sure to tell us what you use this PLC for — we're keen to know!

(160456-I)

COMPONENT LIST

Resistors

Default: 5%, 0.25W

R1,R3,R5,R7,R9,R11,R13,R15 = 100kΩ
R2,R4,R6,R8,R10,R12,R14,R16 = 1kΩ
R17,R18,R19,R20,R21,R22,R23,R24 = 2.2kΩ

Capacitors

C1,C2,C4,C6 = 100nF, 0.2" pitch
C3,C5 = 10µF 50V, 0.1" pitch

Semiconductors

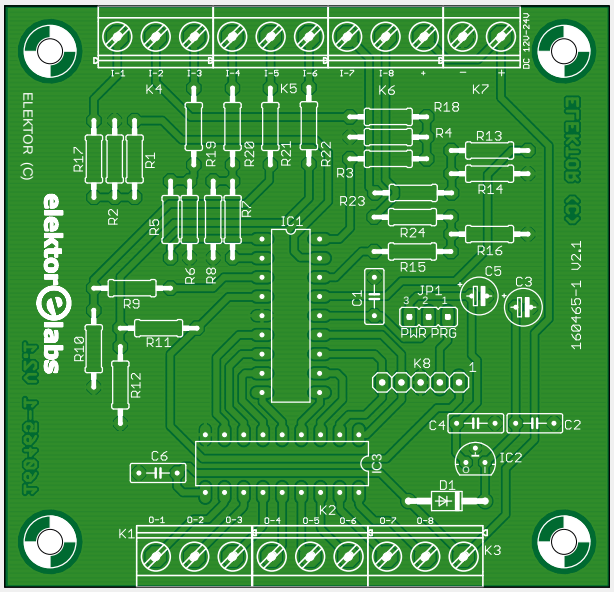
D1 = 1N4004
IC1 = PIC16F628A, programmed, Elektor Store # 160465-41
IC2 = 78L05
IC3 = ULN2803

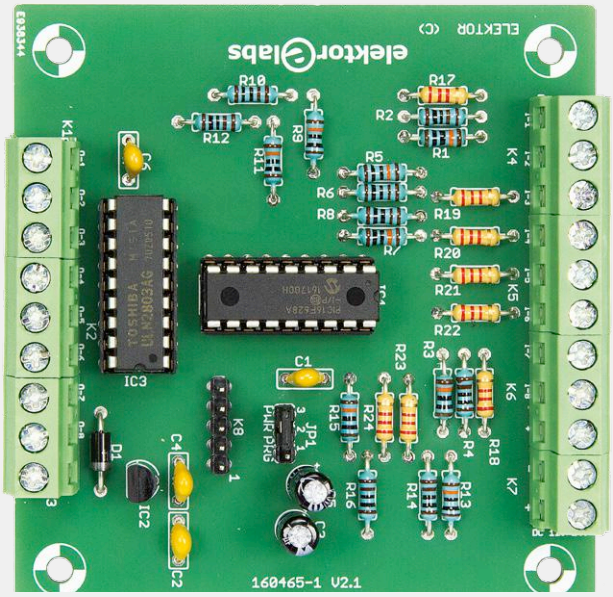
Miscellaneous

K1–K6 = 3-way PCB screw terminal block, 0.2" pitch
K7 = 2-way PCB screw terminal block, 0.2" pitch
K8 = 5-pin pinheader, 0.1" pitch
JP1 = 3-pin pinheader, 0.1" pitch, with matching 2-pin jumper
2x 18-pin DIP IC socket

PCB 160465-1

Figure 7. The PCB for the PLC / traffic signal controller.





Web Links

[1] http://cq.cx/ladder.pl
[2] www.elektormagazine.com/160456
[3] www.elektor.com/tl866a-universal-programmer

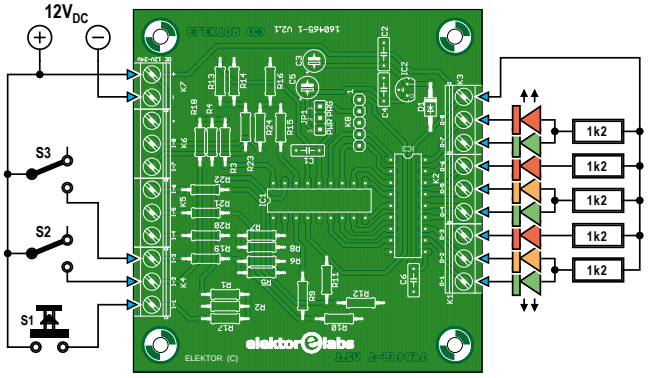


Figure 8. Wiring diagram for the LEDs and switches.

FROM THE STORE

→ 160465-1
PCB

→ 160465-41
Programmed microcontroller