# Ladder Process on Netzer

The upcoming Release 1.5 will have support for custom computations and serial drivers directly executed on Netzer. The choosen programming dialect is Ladder_logic. The IDE for programming will be ldmicro. This article will cover the progress of the implementations. The final feature set is not completed yet.

## Netzer memory organization

### Runtime image

Runtime images are actually interpreted.

The interpreter is entirely written in Assembler for runtime issues.

The reasons for interpretation:

- Easier implementation
- Smaller runtime images
- Security issues (images can be uploaded via Netzer webpage).
- The images can be stored anywhere beyond flash also EEPROM, SD card or SRAM would be possible.

At the moment, the runtime image is stored in a 2K flash memory area in the bootloader. The interpreter images stay persistent even during firmware updates. The syntax (opcodes and parameters) is adopted from ldmicro but is optimized for space and runtime issues.

The translation from ldmicro interpreter code to Netzer opcode files is done via free available IDE extension from MoBaCon (see below).

The translator also integrates some meta data (project name, modification date, found opcodes, ...) which is displayed on Netzer web interface after upload.

The prepared image can be uploaded via Netzer web interface or with command line tools like **curl**. For custom implementations a preloaded image can be integrated in the firmware image.

After loading or when restarting Netzer the image is always checked for validness and consistency.

### Register areas (PABs)

Adopted from ldmicro the runtimes knows about two data types: boolean (Width: Bit) and signed integer (width: 16 bit).

There are two different register areas (called PAB): common PAB and IO PAB.

### Common PAB

This PAB is a real SRAM area in the Netzer space. The size is 256 bytes. Therefore 128 integer or 2048 bit variables can be stored. The common PAB is used as scratch area. The process has exclusive access, no other Netzer software module can change the common PAB. Integer and bit variables share the same address area.

### IO PAB

This PAB is a virtual SRAM area. The IO PAB is divided into integer and bit variables area. Unlike in common PAB integer and bit variables do not share the same address area. For communication between bit and integer areas a copying via the common PAB can be used.

#### Network interface

For communication between the network and the process a special SRAM area is introduced (network variables). There are 8 integers available in each direction (address 0x10-0x17).

*Receiving data from network* Each time the network writes new data, the corresponding mailbox flag in the bit area is set. The process program can poll for this flag and read the value afterwards. Reading clears the flag automatically.

*Sending data to network* Writing to one of the integers sends data to the network. Afterwards the mailbox flag is set automatically. After the network interface has fetched the data, the mailbox flag is cleared. The process should poll mailbox flags to prevent data loss.

#### Bit variables

| Address | Access | Function | Address | Access | Function |
|---|---|---|---|---|---|
| 0x00 | RW | IO0 latch pin | 0x80 | RO | IO0 port pin |
| 0x01 | RW | IO1 latch pin | 0x81 | RO | IO1 port pin |
| 0x02 | RW | IO2 latch pin | 0x82 | RO | IO2 port pin |
| 0x03 | RW | IO3 latch pin | 0x83 | RO | IO3 port pin |
| 0x04 | RW | IO4 latch pin | 0x84 | RO | IO4 port pin |
| 0x05 | RW | IO5 latch pin | 0x85 | RO | IO5 port pin |
| 0x06 | RW | TX latch pin | 0x86 | RO | TX port pin |
| 0x07 | RW | RX latch pin | 0x87 | RO | RX port pin |
| 0x08 | RW | SPI_CS latch pin | 0x88 | RO | SPI_CS port pin |
| 0x09 | RW | SPI_INT latch pin | 0x89 | RO | SPI_INT port pin |
| 0x0A | RW | SPI_CLK latch pin | 0x8A | RO | SPI_CLK port pin |
| 0x0B | RW | SPI_MI latch pin | 0x8B | RO | SPI_MI port pin |
| 0x0C | RW | SPI_MO latch pin | 0x8C | RO | SPI_MO port pin |
| | | | 0x8D | RO | Serial TX FiFo ready |
| | | | 0x8E | RO | Serial RX FiFo data pending |
| | | | 0x8F | RO | RTC time is synchronized |

| Address | Access | Function | Address | Access | Function |
|---|---|---|---|---|---|
| | | | 0x90-0x97 | RO | Mailbox state for incoming network variables |
| | | | 0x98-0x9F | RO | Mailbox state for outgoing network variables |
| | | | 0xA0 | RO | Start flag, TRUE for the first process cycle after start. |

**Integer variables**

| Address | Access | Function |
|---|---|---|
| 0x00 | RW | Latch - All IO port latches in one integer (IO0 is at bit 0 and so on) |
| 0x01 | RW | Edge counter (measured at IO0) |
| 0x02 | RW | Edge counter (measured at IO1) |
| 0x03 | RW | Edge counter (measured at IO2) |
| 0x04 | RW | PWM duty cycle / Impulse width (IO3) |
| 0x05 | RW | PWM duty cycle / Impulse width (SPI_INT) |
| 0x06 | RW | Top of serial FiFo (Reading: RX, Writing: TX) |
| 0x07 | RW | Accessing top of net socket FiFo (Reading: RX, Writing: TX) |
| 0x08 | RW | Reading delivers the current state of Netzer. Writing can be used for executing commands, i.e. restarting. |
| 0x10-0x17 | RW | Network variables |
| 0x80 | RO | Process scratch register. After a division here the modulo result can be found. |
| 0x81 | RO | Ports - All IO port pins in one integer (IO0 is at bit 0 and so on) |
| 0x82 | RO | ADC (IO4) - Measured value as 10 Bit value |
| 0x83 | RO | ADC (IO5) - Measured value as 10 Bit value |
| 0x84 | RO | RTC Seconds |
| 0x85 | RO | RTC Minutes |
| 0x86 | RO | RTC Hours |
| 0x87 | RO | RTC Day of the Week |
| 0x88 | RO | RTC Day |
| 0x89 | RO | RTC Month |
| 0x8A | RO | RTC Year |
| 0x8B | RO | Returns a random value. |
| 0x8C | RO | Edge counter (measured at IO0) - After reading the counter is cleared. |
| 0x8D | RO | Edge counter (measured at IO1) - After reading the counter is cleared. |
| 0x8E | RO | Edge counter (measured at IO2) - After reading the counter is cleared. |

# IDE

The the original ldmicro IDE is located here. MoBaCon has GPL compliant adjusted the IDE for some special Netzer features on base of the official release 2.2. The IDE can can directly generate Netzer bytecode which later can be uploaded via the Netzer web interface.

A patch is posted at the ldmicro forums due the lack of version control - hopefully the patch is integrated in future versions, to downloade it directly from there. As long as this is not possible, we provide a special download here.





LDmicro executable for Windows OS (different languages)

LDmicro source with included Netzer patch (bazaar branch)

Actually all the features of the IDE which can be compiled into interpretable code can be used.

The ADC, PWM and UART stuff is not supported by ldmicro for interpreter targets - but that is no problem because the Netzer solves this via its IO register set.

For that reason a simple naming convention must be considered.

## IO PAB Mapping

Bit variables and integer variables can be mapped directly to IO PAB integer using the @ operator.

To locate integer variable `adc` to the IO location of ADC 4 for example simply rename it to `adc@01`.

To which PAB area (bit or integer) the operator finally maps depends on the variable type.

From:
http://www.mobacon.de/dokuwiki/ - **MoBaCon**

Permanent link:
**http://www.mobacon.de/dokuwiki/doku.php?id=en:netzer:process&rev=1352377836**

Last update: **2025/06/11 20:42**