Machine Automation Tools

Programmable Logic Controller

http://mat.sf.net

Project Leader:	Curt Wuollet (U.S.A.)
Main contributors:	Juan Orozco (Mexico)
	Mário de Sousa (Portugal)
	Jiří Baum (Australia)

`linux.conf.au`
Brisbane, Australia — February 2002

Jiří Baum <jiri@baum.com.au>

Outline

- PLCs and Industrial Automation
  - Specialized computer for industrial control
  - Sometimes connected to expensive or dangerous things
  - Balkanized commercial market
- Overview of the MAT LinuxPLC
  - Goals
  - Architecture
  - Library
  - Modules
  - Programming (in C)
- Hugh Jack's LPC

What is a PLC?

- Programmable Logic Controller
- Developed in the 1960s
- Originally replaced relay-based controls
- Reads input signals from sensors and switches
- Turns outputs on and off based on inputs and logic

Advantages of PLCs over relays

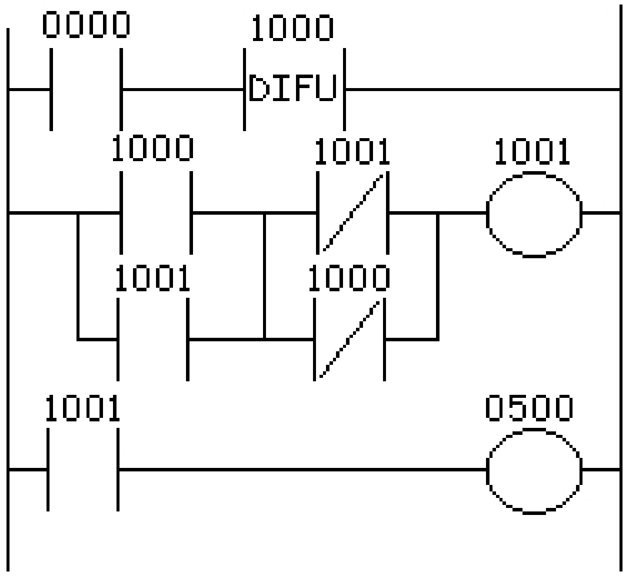
- More flexible and powerful
  - Arithmetics etc
  - Easier to modify logic
  - Networkable
- Simpler implementation (much less wiring)
- Possibly more reliable
- Smaller than relay equivalent

Networkability

- Enables integration of the whole plant
- Better control over whole process
- Centralized reporting, control (SCADA — Supervisory, Control and Data Acquisition), and/or
- Decentralization — logic can be placed closer without sacrificing integration

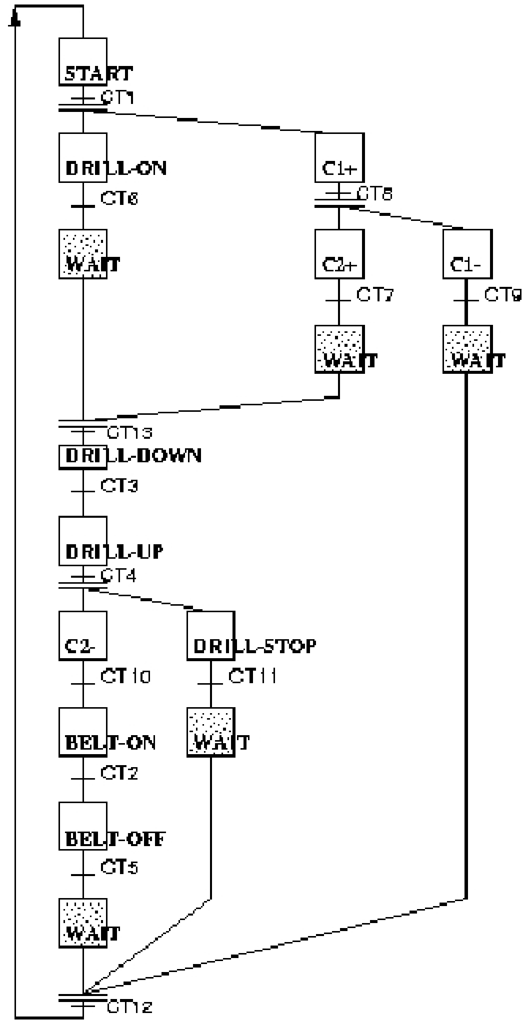
Ladder logic

- Emulates physical relays
  – half-heartedly
- Relay diagram, but
  – Left-to-right, top-to-bottom, cyclic
  – Reads inputs only at top, writes outputs only at bottom
- No data abstraction, very little code abstraction



Other languages

- Statement list
  - Not much higher than ladder
  - Some variants are structured
  - Still little or no data abstraction
- State-based
  - Annotated finite state machine
  - Usually annotated with ladder
  - Often translated to statement list or ladder



Commercial Industrial Control Market

- Balkanized
  - Each vendor has solutions incompatible with all others
    * sometimes with their own
  - A recent standard had 8 different options (IEC 61158)
    * not counting vendors' variants...
- Much inefficiency and duplication
- Tower of Babel

Tower of Babel

- A few examples
  - BACnet
  - CAN
  - ControlNet
  - DeviceNet
  - EIA 709.1 (Lonworks)
  - Fieldbus (various)
  - IEC61850
  - Manufacturing Automation Protocol (MAP)
  - Modbus
  - OPC (based on Microsoft's DCOM)
  - Profibus
  - SPI

  ⋮

SoftPLC

- The software version of the PLC
- Runs on a PC, does the job of a PLC
- Commercial example — AutomationX
- Properly, that's what MAT LinuxPLC is
  - Or, at least, that's one use
- Pros and cons
  - Graphical screen included by default
  - More flexible / more complex programming
  - Typically less rugged / reliable

Timing

- Significant stated concern
- PLC cycle usu. measured in ms
- Regularity and predictability more important than speed
  - Timing glitches hard to debug

Reliability

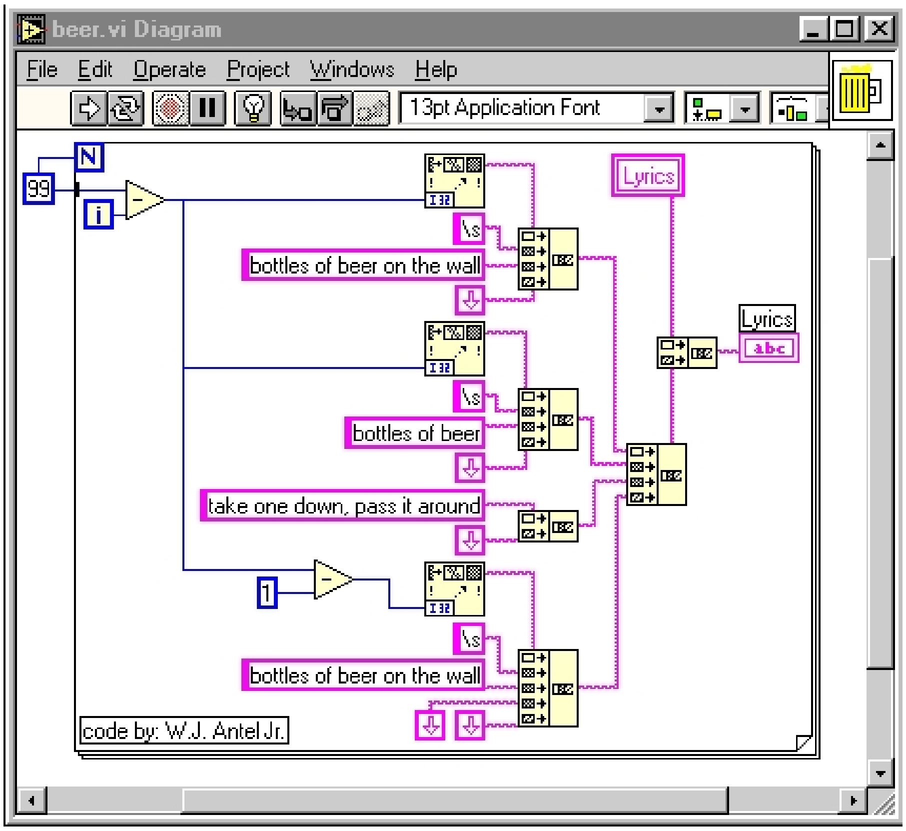
- Failure is stated "not an option"
- Obviously, things still fail (valves stick etc)
- Downtime can be very expensive
- Sometimes directly affects safety
- Preferably not at 3am (Murphy's law)

Maintenance

- Accessibility to personnel
  - Often non-programming, hence the ladder
- Long-term availability of spares
  - Preferably same version number
    - Advantage of Open Source
- 3am, expensive downtime

PC-based control — Labview-style

- Graphical-based programming (connect-a-box)
- Little scope for abstraction
- Data-flow, but imperative under a thin skin



- AutomationX another example

PC-based control — custom program

- Control program written in a general purpose language
  - C
  - Visual Basic
  - Delphi
- Lot of work
- Not particularly suitable
  - Parallelism
  - Jumping in and out of sequence
  - Reliability
    * OS, programming language not designed for purpose

Two-layer system

- A two-part system
  - PLC for reliability
  - PC for flexibility
- PLC wired to PC's reset or power

H/W interfacing

- Cards in PC
  - With external isolation board
- RS232 or bus to a dumb (or smart) PLC
  - via one of the many different busses
- Ready-made embedded boxes

Embedded linux boxes

- Beginning to appear
- Supplied with linux as standard
- Might run our s/w
- Simplifies use
- Avoids impression of PC-ness

Microsoft

- Programming tools and uploaders
  - often only available for MS
- Sometimes MS Windows box in control directly
  - security
  - reliability
  - long-term availability
- Legacy of impression of unreliability

PLC — summary

- Specialized computer for industrial control
- More flexible and powerful than relays
- Can be connected to expensive or dangerous things $\rightarrow$ reliability required
- Balkanized commercial market
- SoftPLC is a PLC running on a generic PC

Why MAT LinuxPLC

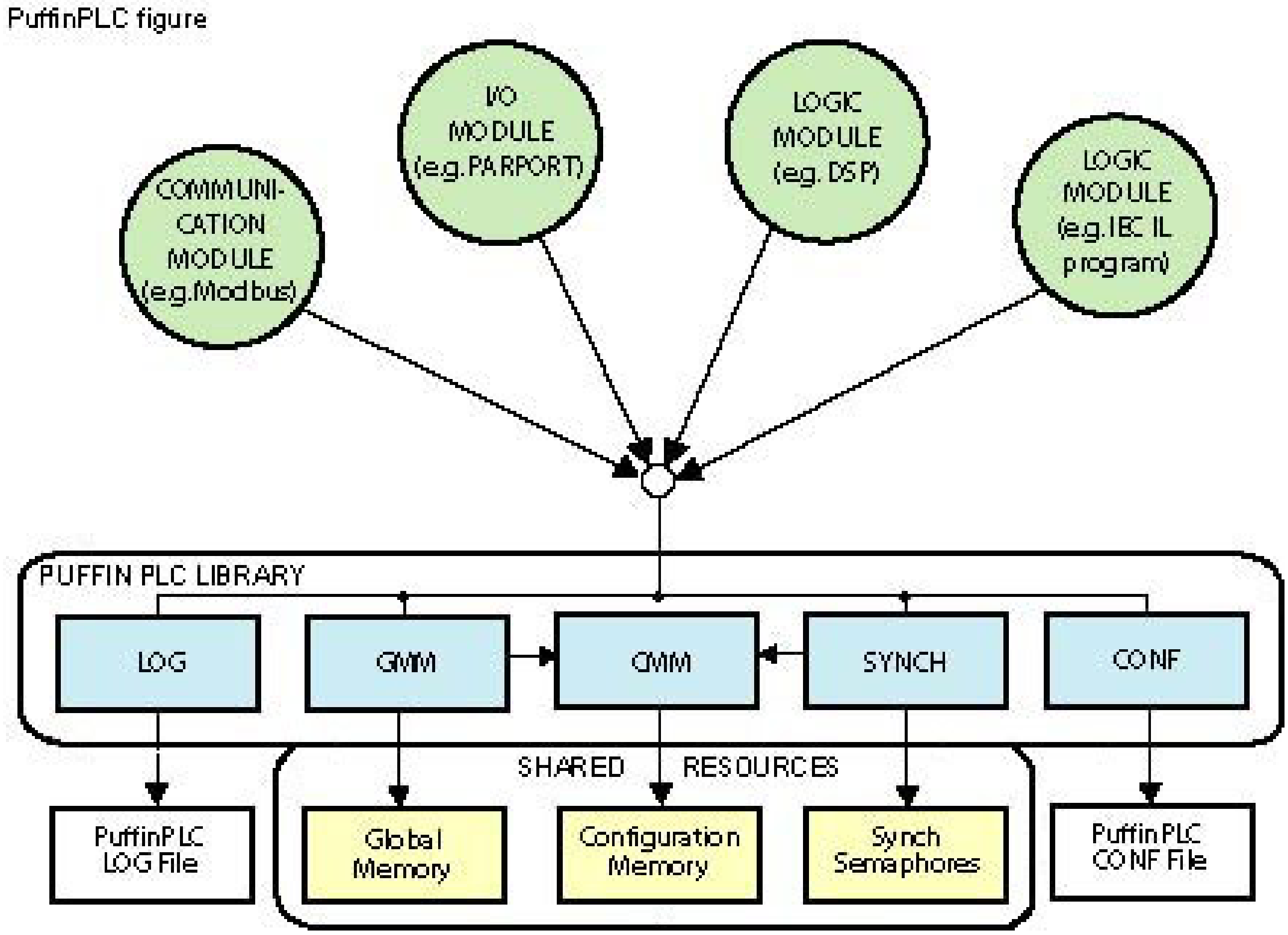
- Truly open platform for automation
  - Helps avoid lock-in
  - Facilitates integration of disparate systems
  - More flexible than closed-source SoftPLC
  - Gives power to integrator and end-user
- Engender cooperation and interoperability
- Avoid duplication of effort

MAT LinuxPLC — technical details

- Architecture
- Modules
- Library
- Programming (in C)
- Hugh Jack's LPC
- Summary and to-do list

MAT LinuxPLC Architecture

- Highly modular
  - Modules are separate processes
  - Shared library (`.so`)
  - Shared memory, protected by a semaphore



Top-level setup

- `linuxplc.conf` specifies
  - Modules to run
  - Points (inter-module variables)
  - Parameters for modules
  - Other files containing info
- Some modules need compilation / translation before starting (use `make`)

Module groups

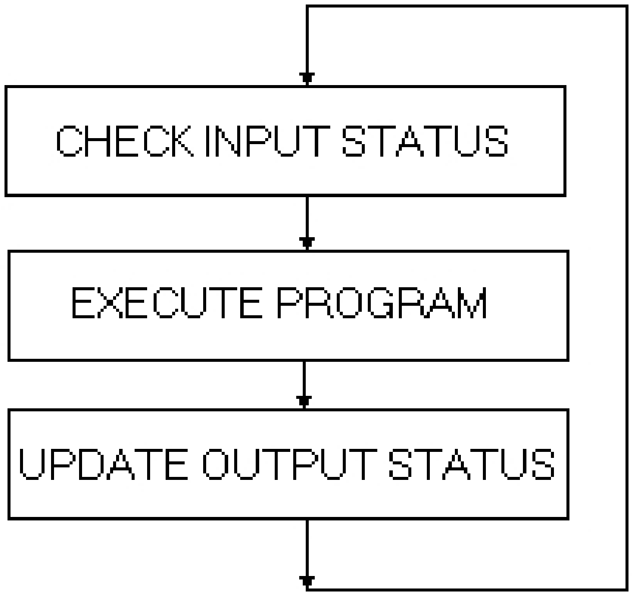
- Modules divided into three broad groups
  - Logic engines — the actual calculations and decisions
  - I/O and comms — contact with the Real World
    - input signals from sensors and switches
    - outputs to motors, hydraulics, pneumatics, lights etc
  - Human-Machine Interface — interaction with humans (screen display, keyboard/mouse)
    - control panels (buttons, lights etc) go through normal I/O, not HMI
- Orientation only, no intrinsic difference

Module types

- Generic — used directly as they come
  - eg DSP, classicladder, most I/O
- Specific — generated by tools included
  - eg the mnemonics translator
  - need to be compiled before starting (`make`)
- Custom-made — written directly in C

Execution model

- Most modules run an infinite loop
  - PLC-style

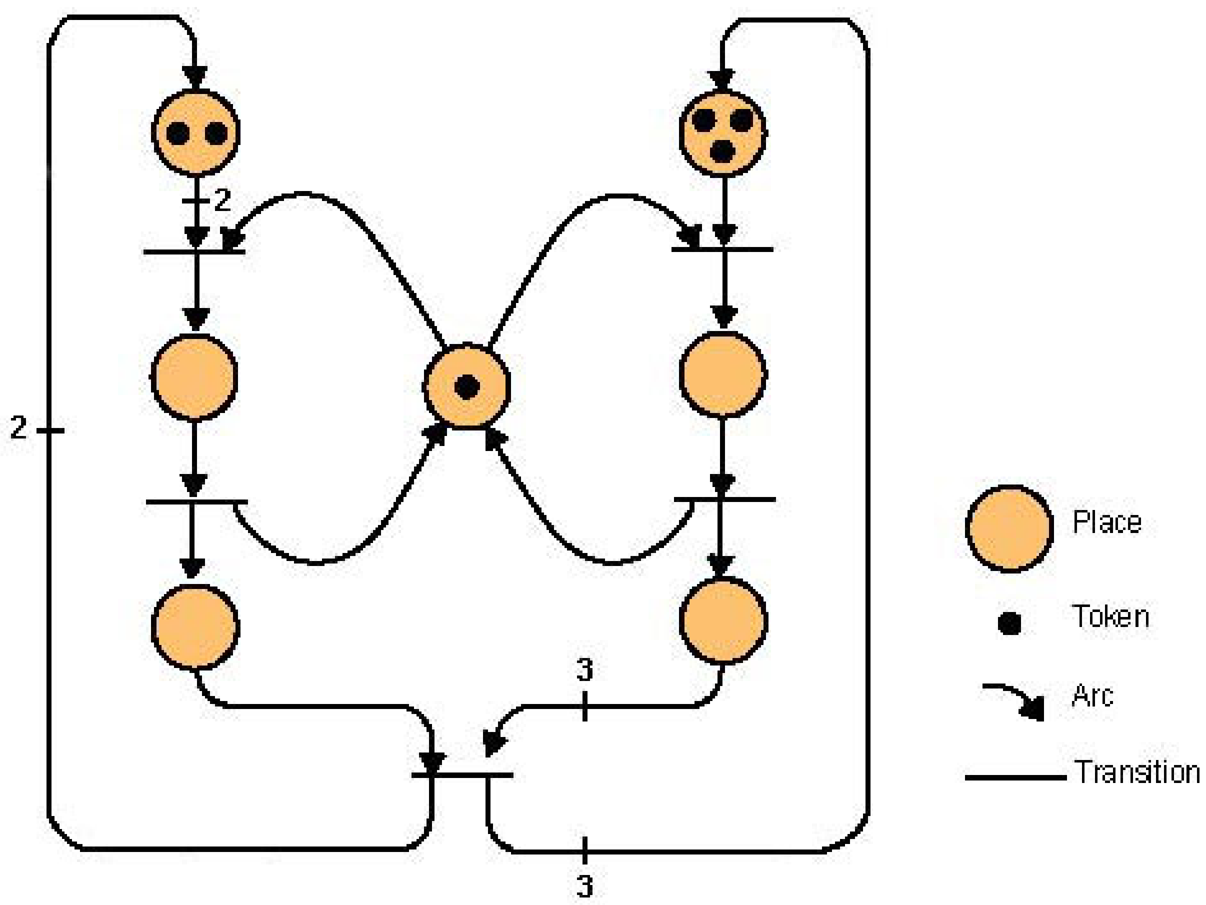


  - up to the module
- Default asynchronous, each at its own speed
  - semaphore gives atomic updates
- Additional synchronization optional
  - But must either synch or set scan period or be I/O bound, otherwise hogs CPU
    - ∗ linux scheduling issue — may be better with new kernels

Synch

- Petri net — or simplified syntax
- Synch points
  - Normally at top and bottom of loop
  - May have more

Petri Net figure



Global map

- All data in ‘points’, in a global map in shared memory
- Each point 0-32 bits (typically 1 bit)
- No distinction between input, output, internal
- Each point writeable by exactly one module
- A point may be both input and output
  - ‘traffic cop’ with no logic module

Global map — updates

- PLC-like semantics for updates
  - Global map updated only at end of scan
  - Buffered in local map in each module
  - Semaphore for atomic updates
  - PLC skills transfer over
- Optionally isolated for stray pointer protection

Shared library

- GMM (global map)
- CMM (config map)
- config (parses config)
- synch (provides synchronization)
- period (set scan period)
- state (manage module state)
- log (uniform logging)

I/O modules

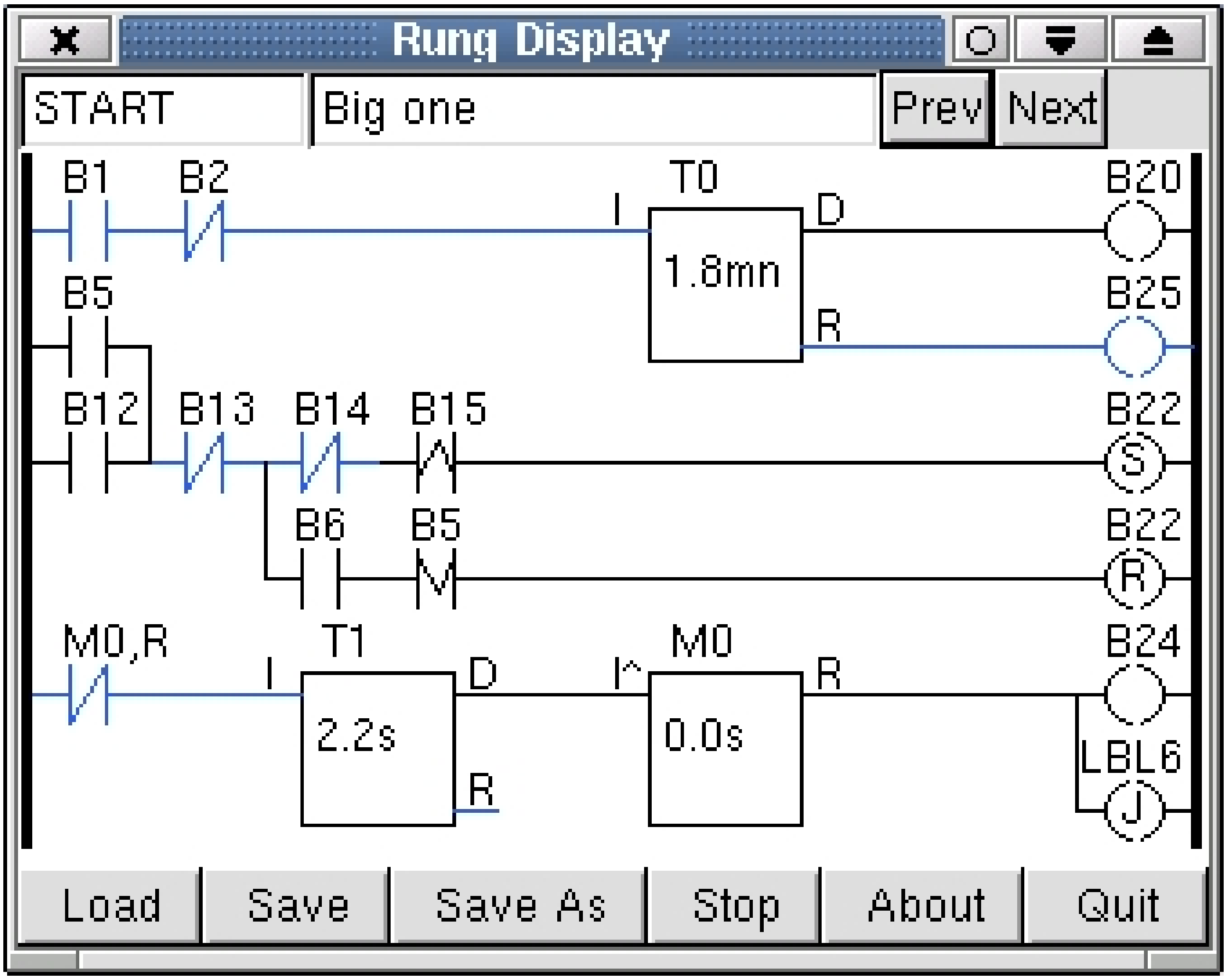
- Somewhat lacking...
- 8255 chip — most cards use this
- Modbus — one of the bus protocols
- CIF — a card from Hilscher
- Parallel port direct access

Logic engines

- Ladder — graphical — Marc Le Dourain's 'classicladder' project
- Ladder — mnemonics — a simple Perl script
- DSP (Digital Signal Processing)
  - typeconv, add, mult, pow, filter, ramp, nonlinear, alarm, multiplexor
  - PID

classicladder

- Traditional stepladder
- GUI-only
- (Lacks a non-graphical executive)
- Nicely animated for ease of debugging

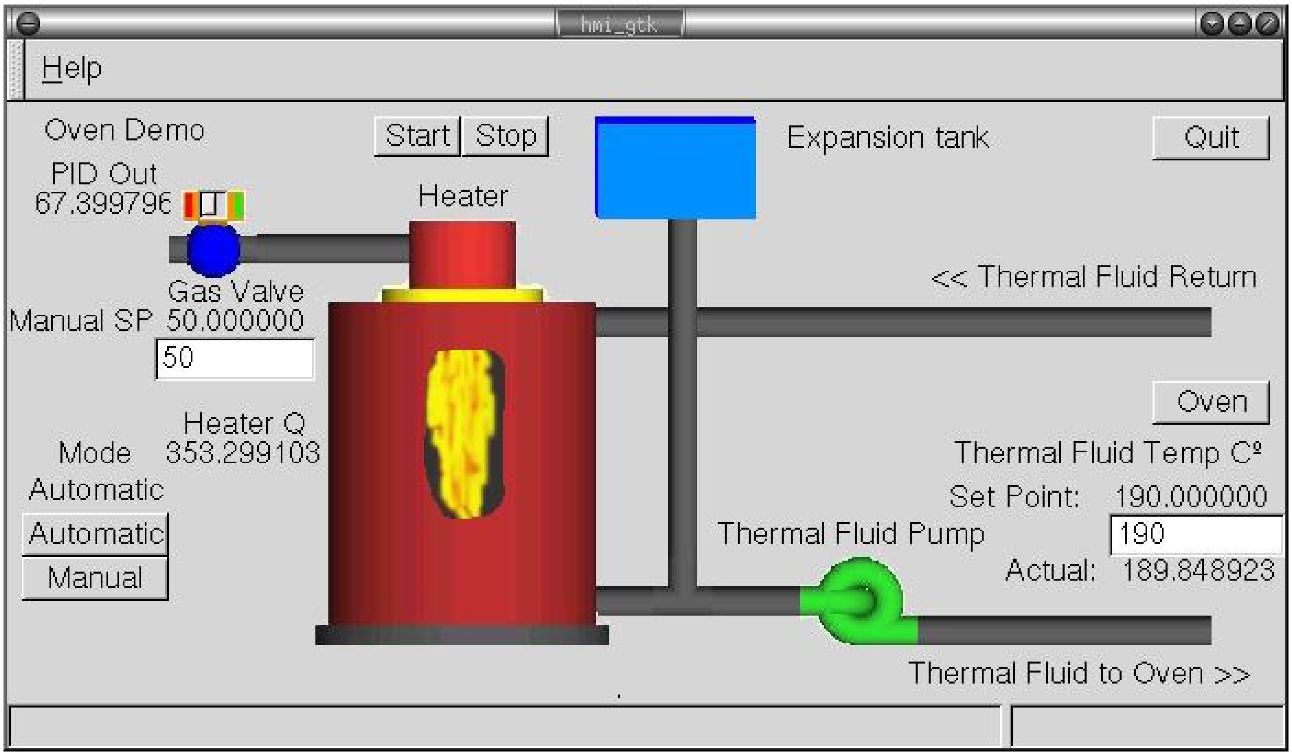


DSP

- Sequence of blocks
  - typeconv — convert to and from the float the other blocks use
  - add, mult, pow — standard mathematical functions
  - filter, ramp, nonlinear — misc filters and transforms
  - alarm, multiplexor — comparison, selection
  - PID — Proportional-Integral-Derivative closed-loop control

Human-Machine Interface

- Gtk
  - Glade for layout



- Text-based (curses, kbd)
  - Sometimes X is over-kill and imposes load on the CPU, disk, memory etc

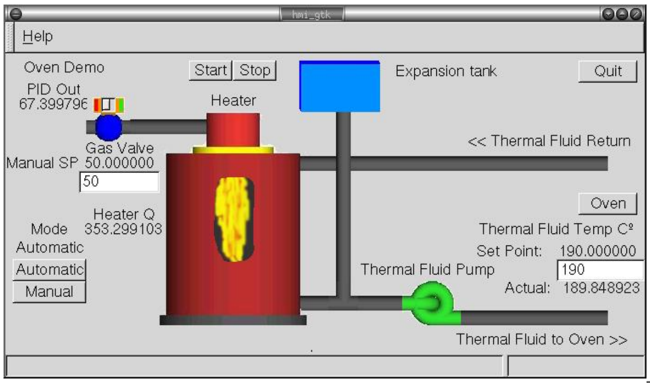
Simple Demo

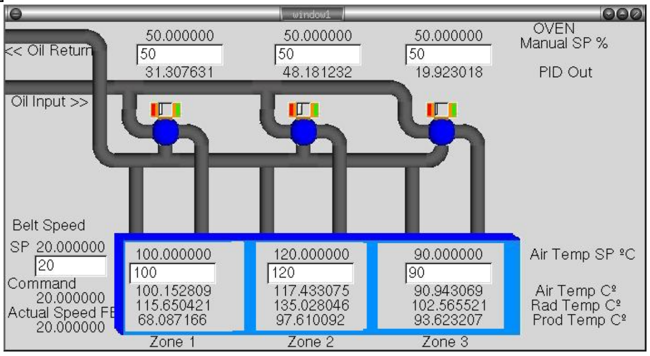
- Simple light-chasing — the equivalent of ‘hello world’
- Modules:
  - Keyboard input (linux console or stdin)
  - Chaser (written in C)
  - Curses display
  - Shutdown

```
                    Light chaser
                    ============

                       Lights:

                     1  2  3  4
                     -  *  -  -

                       Buttons

                         <->


   This is a simple light-chasing demo, controlled from the keyboard. By
intention, this demo requires no hardware or setup - not even X or svgalib.
 That's why it looks the way it does - see the other demo directories for
                     more impressive demonstrations...

                      Controls (from the keyboard)
                      ----------------------------
      - L and R - toggle the ‘left’ and ‘right’ points. Remember to
        toggle them off again before trying the other direction :-)
      - Q - toggles the ‘quit’ point, which terminates the demo.
```

Oven Demo

- Simulates a simple continuous process
- Manual/automatic mode, setpoints
- Modules:
  - DSP for simulation of heater
  - DSP for oven control
  - Gtk gui (or curses)
  - Shutdown





Typical module

- Algorithm:

```
plc_init(...)
grab handles to points
while(1)
   plc_scan_beg()
   plc_update()
   logic
   plc_update()
   plc_scan_end()
```

Initialization

- `plc_init(...)` initializes the library
- Get handles to points
  - Look up in config
  - Check access mode
  - Make access real-time
- Anything else from the config file
- Any other init (eg hardware for I/O modules)

Main Loop

- `while(1)`
  ```
  plc_scan_beg()
  plc_update()
  ```
  logic
  ```
  plc_update()
  plc_scan_end()
  ```

- `plc_scan_beg()` and `plc_scan_end()` contain default synch points, scan period, etc
- `plc_update()` updates the local and global maps
- Logic uses `plc_get(`*point*`)` and `plc_set(`*point*, *value*`)` to get and set points

I/O module library

- Simplifies and streamlines writing
- Makes config of I/O more uniform
- Includes main() and the entire framework
- Only the actual h/w specific parts need be written

Hugh Jack's LPC

- Linux Programmable Controller
- Similar project
  - Same goals
  - Highly modular
  - etc
- Plan to merge

Hugh Jack's architecture

- Modules
  - Run as threads
  - Dynamically loaded
- Interact via
  - Global variables (similar to MAT points)
  - Local/remote variables
  - Messages

Hugh Jack's modules

- Ladder logic
- Telnet server and java clients
  - Monitor
  - HMI
  - Ladder programming
- Serial port
- Devicenet
- DAS08 board (I/O)
- E-mail sending

MAT — summary

- SoftPLC for Linux
- Modular architecture
- Work is in progress
  - Infrastructure is in place
  - Modules being written

MAT To do list

- Documentation
- Merging with LPC
- More I/O modules
- More/better logic and HMI modules
- Take over the world