YRmx

Case study:
a Real-Time Framework optimized for Process Control in a family of industrial equipments

Giuliano Colla
Copeca srl.

COPECA

Industrial process

- A complex process involving many simultaneous operations can be split in a number of almost-independent sub-processes
- The sub-processes can be implemented as tasks in a real-time multitasking environment

Linux real-time

- Modern Linux vanilla kernels provide latencies appropriate for not too demanding soft real-time multitasking
- RT_PREEMP kernels make possible user-space hard real-time multitasking

Process Task

- **A task is neither a Unix process nor a Unix thread**
- It is better defined as a sub-process
- It has independent life like a process
- It takes advantage of the thread scheduling mechanism
- It shares the environment with other tasks like a thread

Implementing process Tasks

- Can be implemented as threads, but keeping in mind the task peculiarities
- **An appropriate framework can provide appropriate API's, making development easier, faster and less error-prone**

YRmx framework

- Provides a Task software object, by encapsulating a Linux thread
- Provides a message-exchange mechanism for:
  - task synchronization
  - inter-task communication
  - delay
  - periodic execution
  - mutual exclusion and memory management

COPECA

YRmx mechanism send message

- A task sends a message to an exchange:
  - If another task is waiting at the exchange, it receives the message and is made runnable
  - If no task is waiting, the message is appended to the exchange

YRmx mechanism receive message

- A task waits for a message at an exchange:
  - If a message is available, the task receives the message
  - If no message is available, the task is suspended (blocked)

YRmx mechanism timing

- A task waits for a message at an exchange and specifies a timeout value:
  - If a message becomes available, before the timeout, the task receives the message, and becomes runnable
  - If no message arrives, when the timeout expires the task receives a **timeout_type** message and becomes runnable

YRmx mechanism response_exchange

- Each message includes a *response_exchange* field to provide a two-way communication

YRmx features

- Ease and simplicity
  - Reduced training time
  - Reduced probability of errors

COPECA

YRmx features

- Task isolation
  - Parallel development: reduced time-to market
  - Independent test and simulation: better robustness with lower development costs

COPECA

YRmx features

- Contextual activation and communication
  - Lower latency
  - Reduced risks of improper handling of resource contention

YRmx features

- *response_exchange*
  - Two way communication
  - Eliminates the need for call-back

COPECA

Frank program test

I7 4 core @1.2 GHz		Min	Average	Max
sem_timedwait				
	One-way	2158	16355	45616
	Round trip	3458	**21184**	50379
Delay list				
	One-way	1761	6869	37505
	Round trip	3234	**11287**	42658

Case study: Bookmaker machines

Bookmaker machines

COPECA

Bookmaker machines

COPECA

Specifications

- 2000 booklets of 25 pages/hour
- Over 50,000 pages/hour
- Guarantee booklet integrity and security
- User friendly GUI
- Store and provide information for QC, traceability and usage optimization
- Connect to factory LAN
- Connect from remote

COPECA

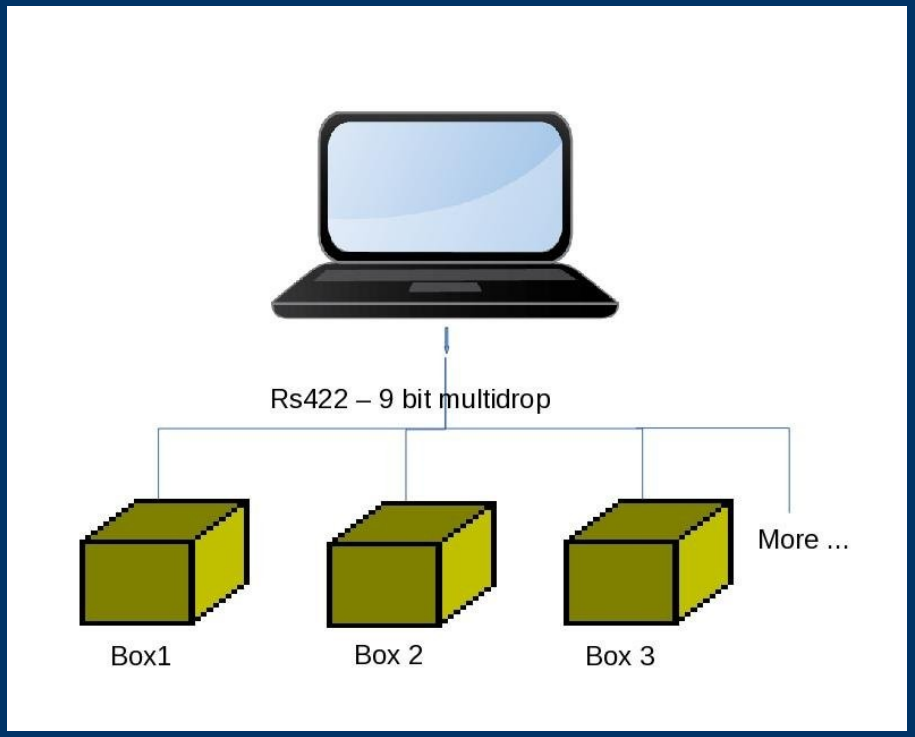
Project data

- The operations to fabricate a booklet require 20 to 25 seconds
- Fabrication must be split in steps to be executed in parallel
- From 50 to > 100 sub-processes/tasks
- 1 ms lost for each page determines a throughput loss of 25 booklets/hour

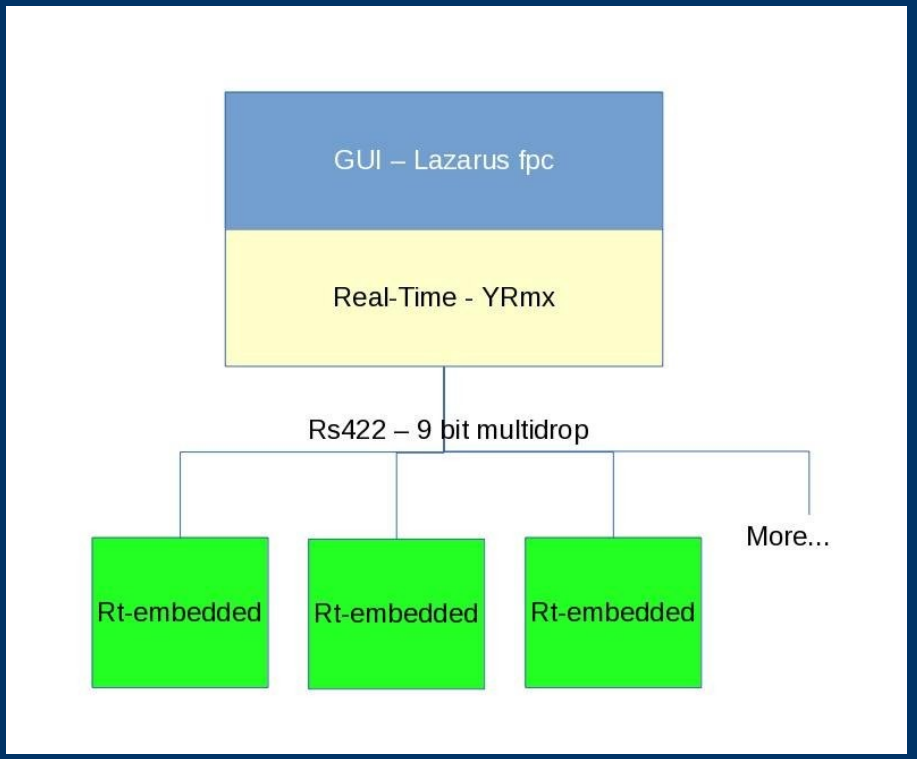
COPECA

System design

Hardware Layout



Software Layout



COPECA

Hardware requirements

- Harsh industrial environment
- Long term support
- Good Linux support
- Good quality TS display

COPECA

Hardware selected

- Industrial grade
- 10 Years support
- Intel I7 quad core @1.2 to 2.2 GHz
- 16GB Flash disk
- 15” TFT with TS
- NCS Computers

COPECA

System software

- CentOS 6 (RHEL 6)
- Kernel 3.10.10-rt7
- KDE Desktop (Qt based)
- Fpc Lazarus – Object Pascal – Qt widgetset

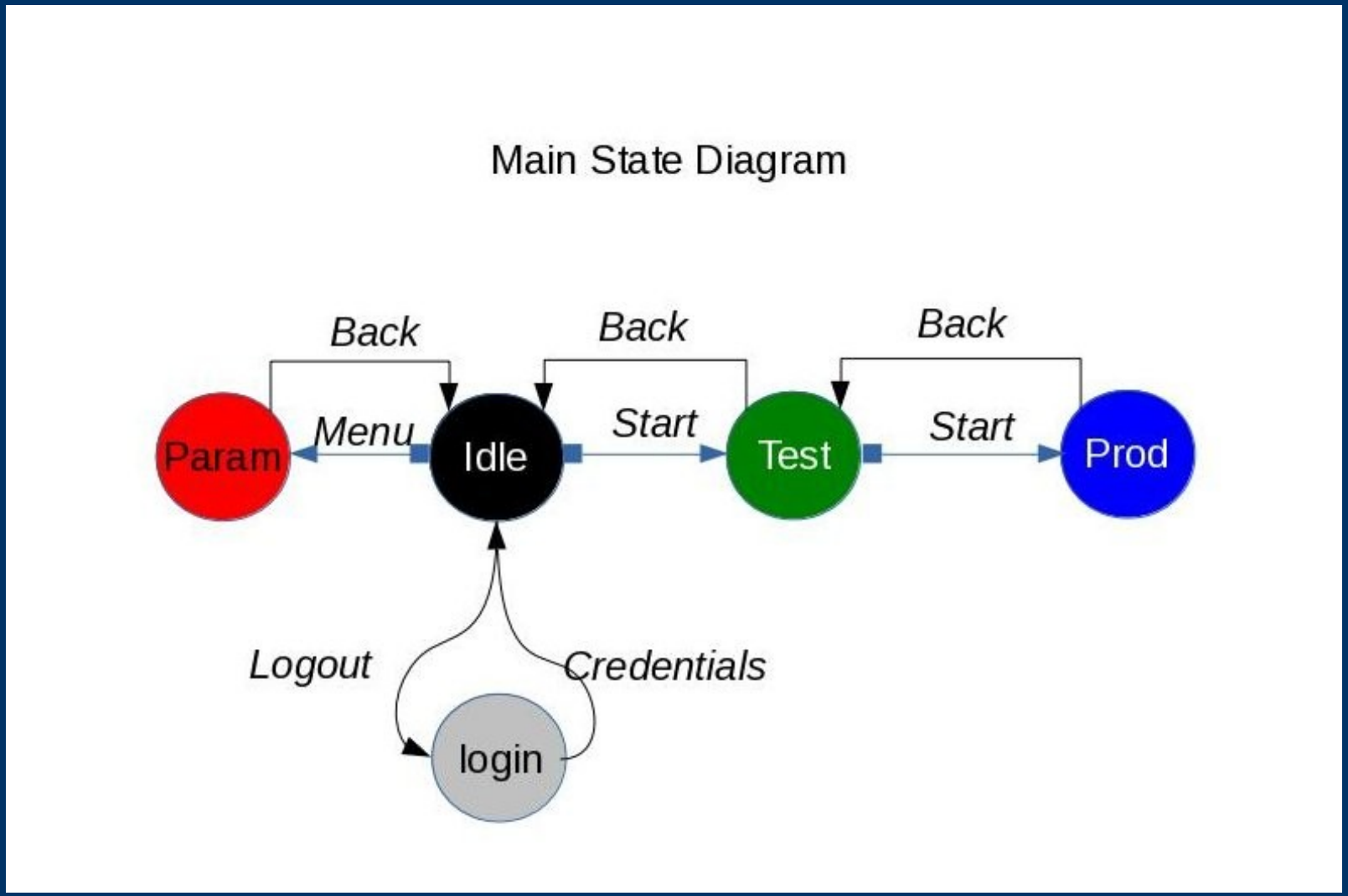
COPECA

Human Interface requirements

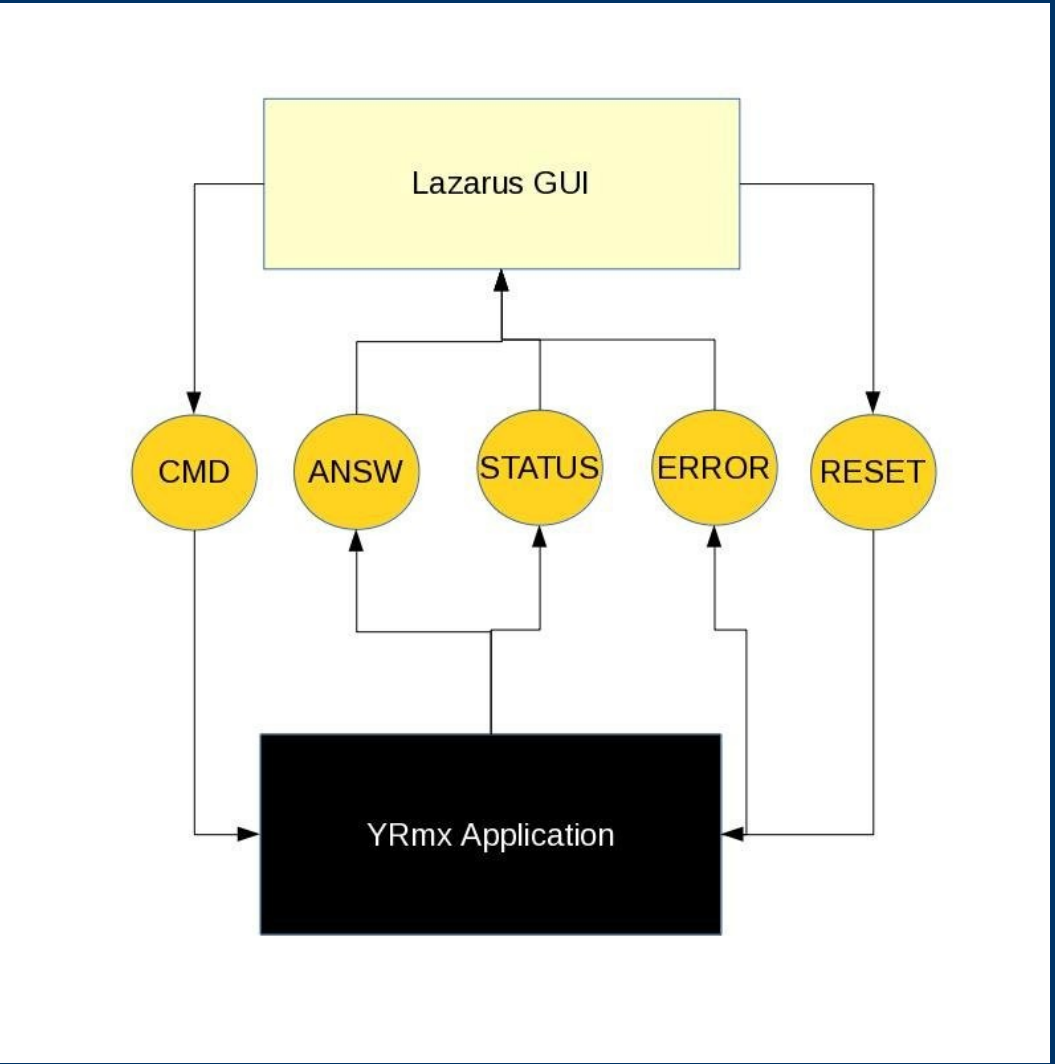
- Pleasant look
- User friendly
- Must interface seamlessly to Real-Time tasks for
  - Sending commands
  - Receiving responses
  - Reacting to asynchronous events
- Must provide different levels of permissions
- **Must hide whatever is not allowed in the current context**

COPECA

Main state diagram



IPC layout



COPECA

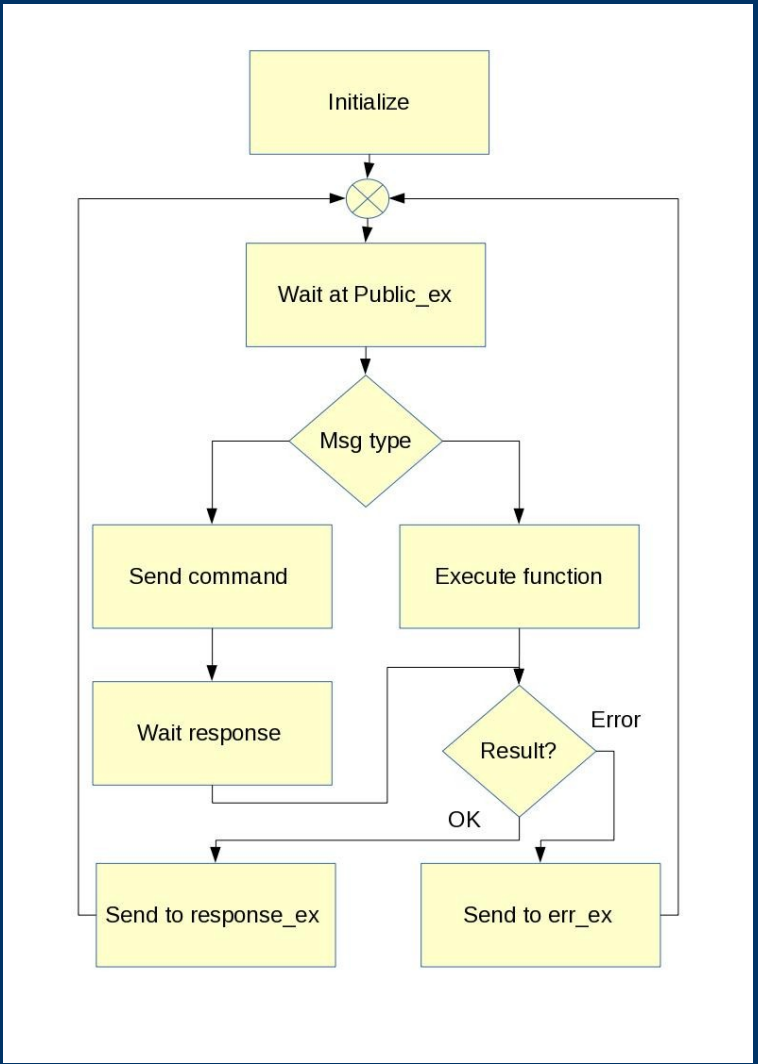
Real-Time design

- Keep private all exchanges not needed by other tasks
- All application messages are of the same length
- A task receiving a message must always modify appropriately content and *type* and
  - Either it forward to another exchange
  - Or send it back to the *response_exchange*

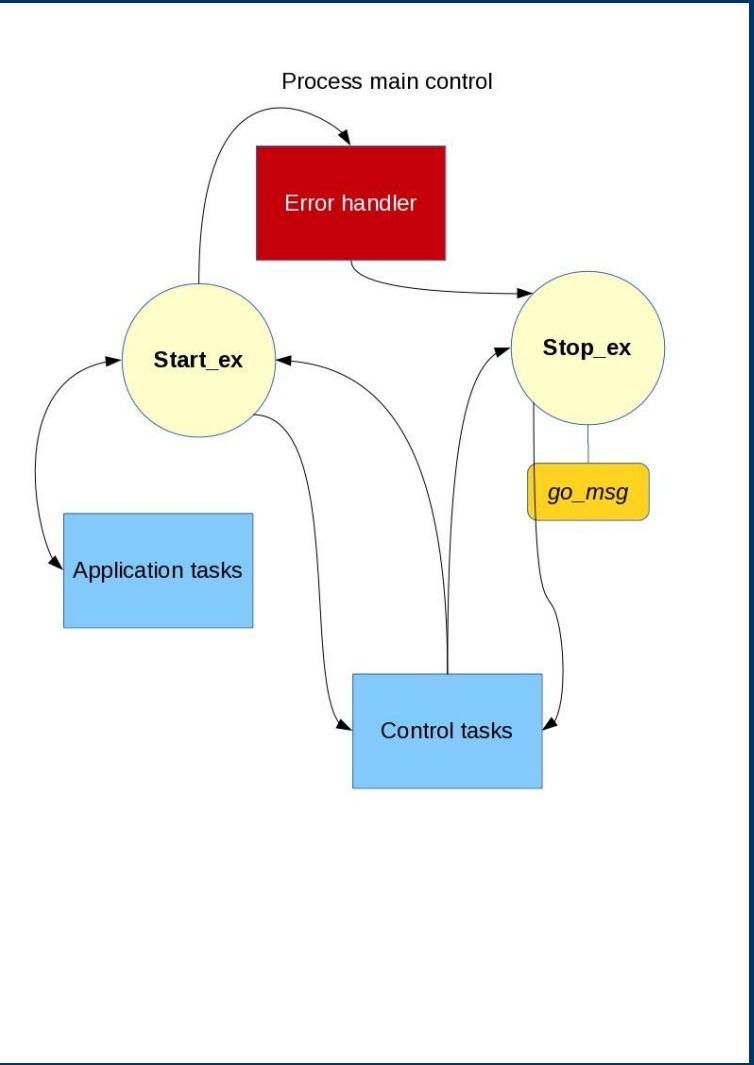
Message response rules

- A response must specify if the requested action was executed or not.
- A response must specify if an error was encountered (odd *type*) or not (even *type*)
- A non-error response must be sent to the *response_exchange*, an error response must be forwarded to the error exchange (*err_ex*)

Skeleton application task



Main process control



State Machine: Running – Stopped

COPECA

Process control

Simple synchronization

```
static void WaitOk (void) {
MSG_DESCRIPTOR *hmsg;
    hmsg = rqwait(&start_ex,0);
    rqsend(&start_ex,hmsg);
    }
```

Support for step-by-step

```
static void WaitOk (void) {
MSG_DESCRIPTOR *hmsg;
    hmsg = rqwait(&start_ex,0);
    if (ByStep)
       hmsg->response_exchange = &stop_ex;
    rqsend(hmsg->response_exchange,hmsg);
    }
```

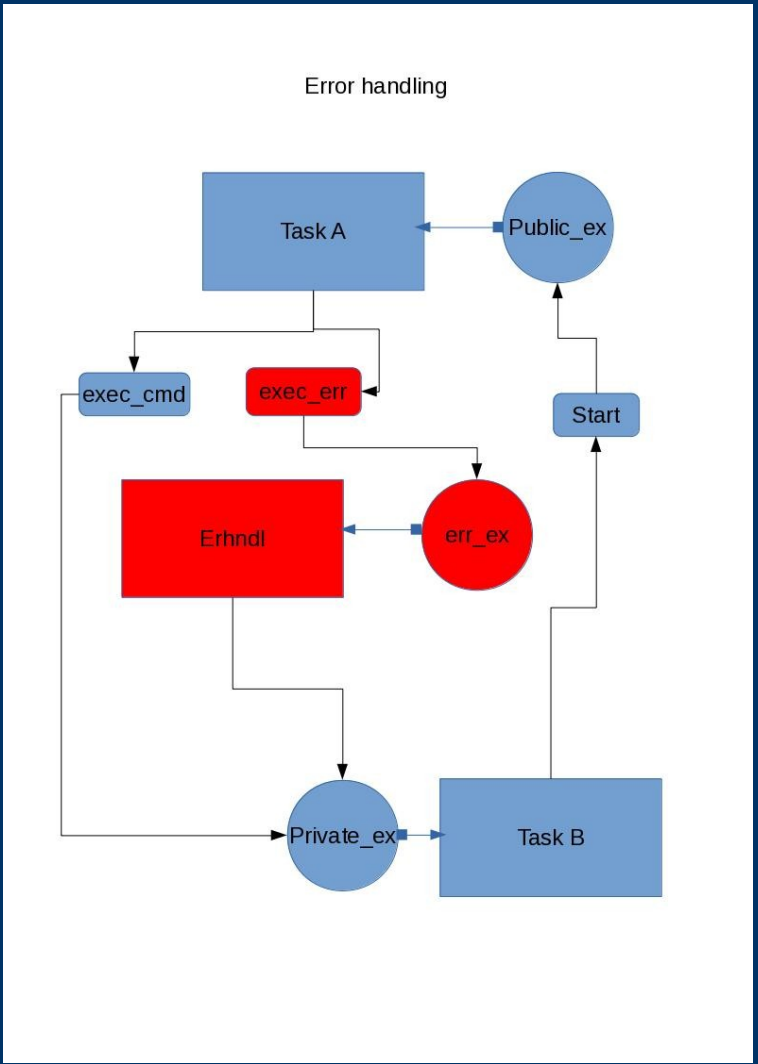
COPECA

Search go_msg

```
static MSG_DESCRIPTOR *search_go (void) {
    MSG_DESCRIPTOR *go_msg;
    go_msg = rqacpt(&start_ex);
    if (go_msg == NULL) go_msg = rqacpt(&stop_ex);
    while (go_msg == NULL) {
        go_msg = rqwait(&start_ex,5);
        if (go_msg->type == timeout_type) {
            go_msg = rqwait(&stop_ex,5);
            if (go_msg->type == timeout_type)
              go_msg = NULL;
            }
        }
    return (go_msg);
    }
```

COPECA

Error handling



COPECA

Serial communication

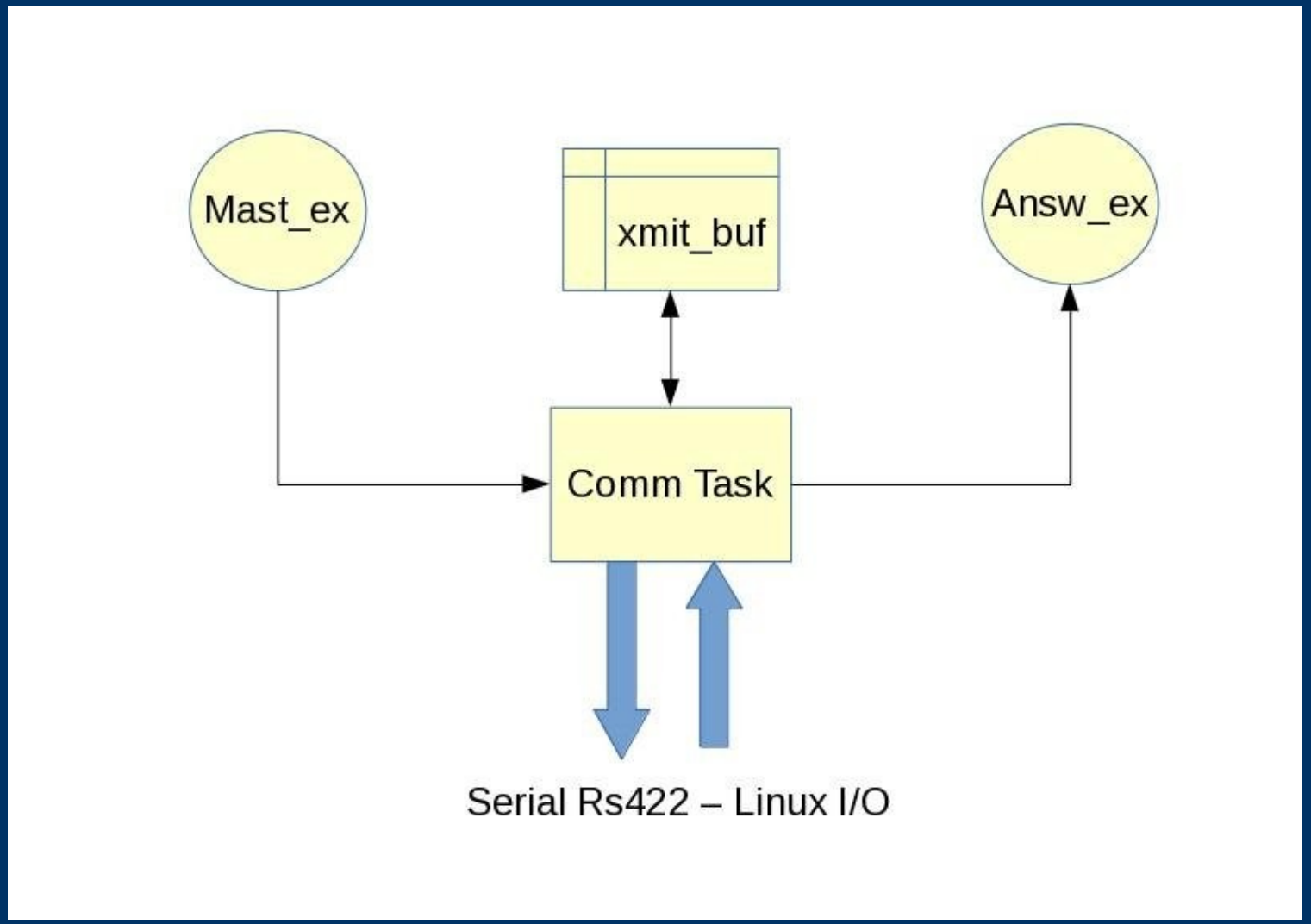
- BitBus derived protocol
- Standard Linux driver
- 4 wires RS422
- 9 Bit multidrop
- 230 kBaud

COPECA

Transport layer

- Keeps a buffer for each slave
- Poll slaves at 1 ms rate
- Receives packets to send at *mast_ex*
- Sends transmission results to individual box appropriate exchange
- Sends received packets to *answ_ex*

Transport layer (simplified)



COPECA

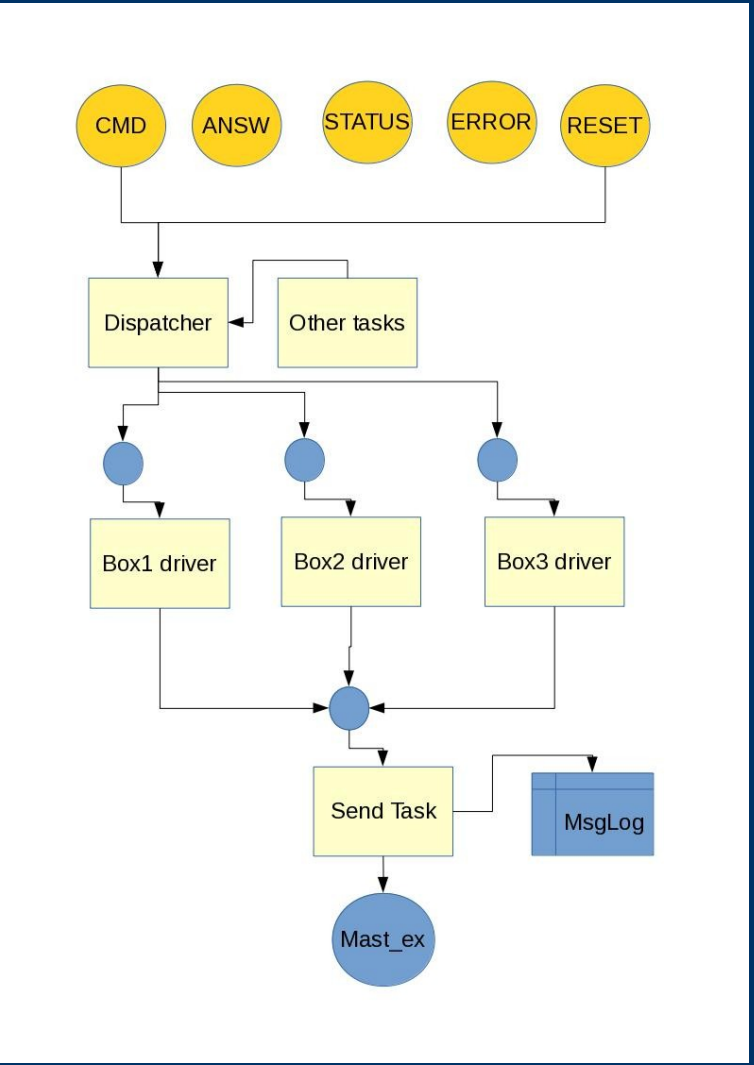
Data transmission specs

- Packets as short as possible (each byte requires 50µs)
- Handle simultaneous requests for send to different slaves
- Perform message-to-packet convert
- Message sent only when ACK received
- Handle response to a message
- Handle send failures

Data transmission

- Strip the header, leaving only tag and type
- Split the logic into a driver task and a send task
- One driver task per slave
- Perform message-to-packet convert
- Maintain a table of sent messages to match answers (conntrack-like philosophy)
- Handles an abort requests

Data Transmission (simplified)

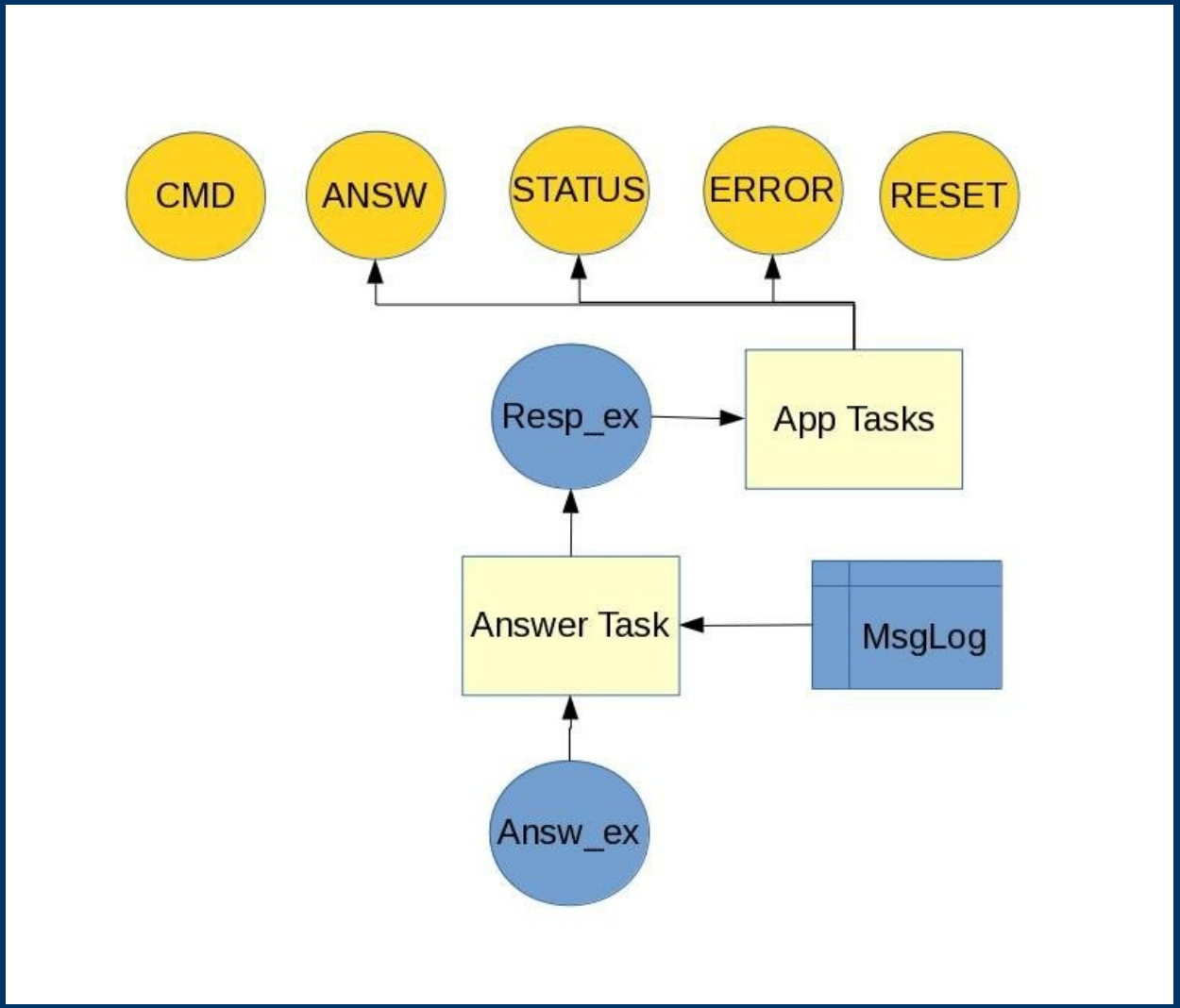


COPECA

Data receive

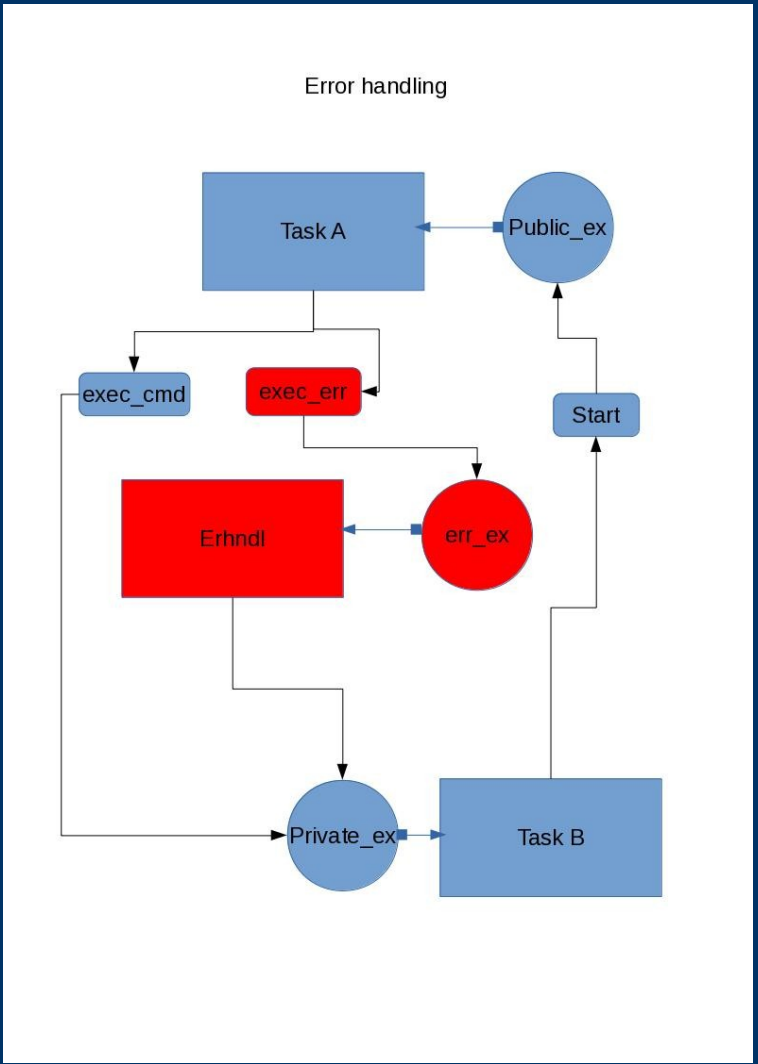
- A received message tag field can be
  - Non-null: response to a message sent
  - Null: asynchronous (unsolicited) message for status or error information
- Valid tag field: the original message is found in the table, and modified with received data
- Null tag field: a message is taken from the pool and modified with received data. The *response_exchange* is taken from a dispatch table from the *dev* field.
- The resulting message is sent either to the *response_exchange* or to *err_ex*

Data receive (simplified)



COPECA

Error handling

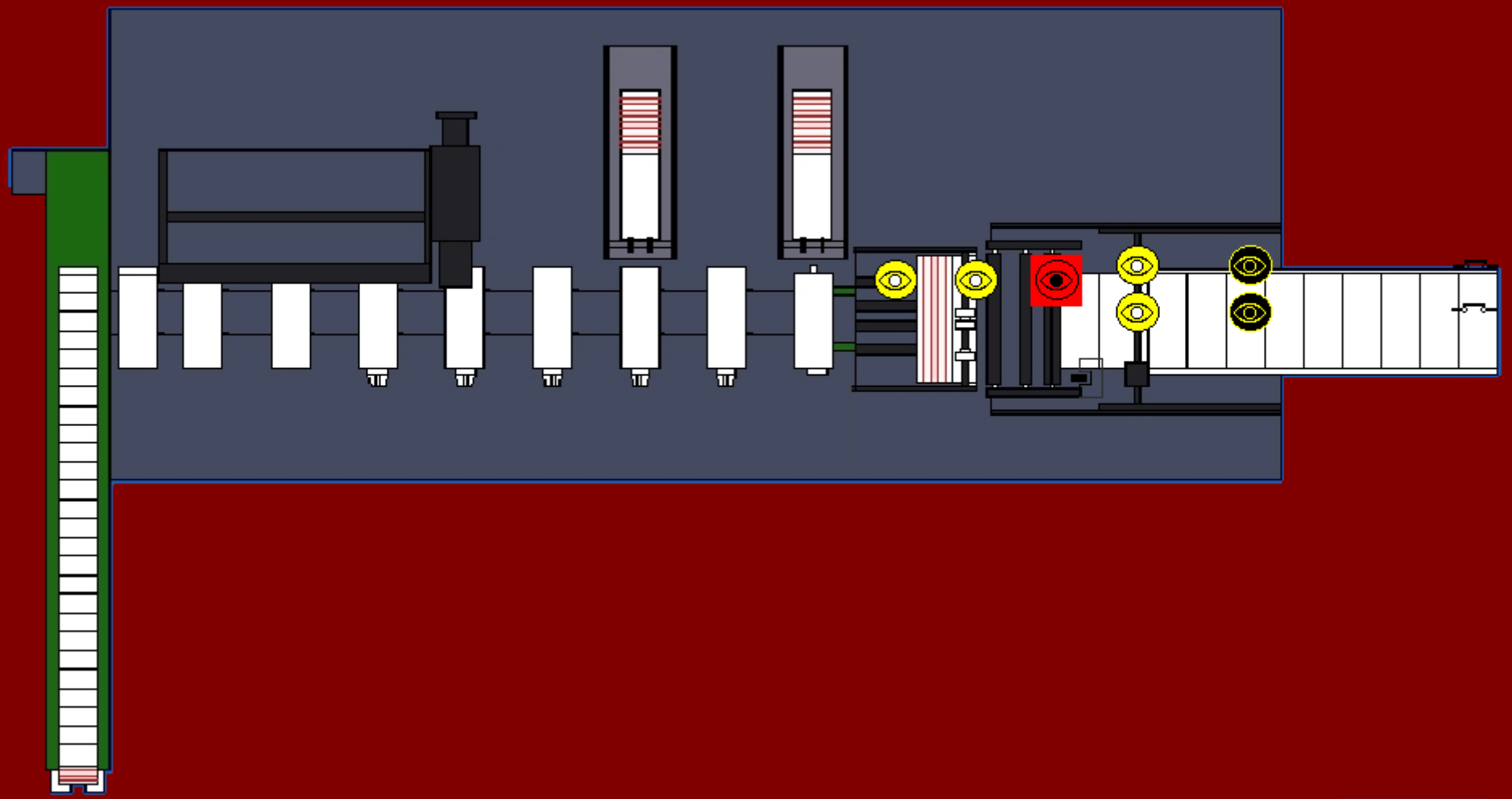


COPECA

Message structure

```
#pragma pack(1)
#define MaxPosLen   192
typedef struct {
        msghdr;
        unsigned short  tag;
        unsigned char   unit;
        unsigned char   len;
        unsigned char   device;
        unsigned int    fstat; ← Sensor state
        unsigned int    ftout; ← Timeouts
        unsigned int    fover; ← Overruns
        unsigned short  cuts;
        unsigned char   pos[MaxPosLen];
        } STANDARD_MSG_DESCRIPTOR;
```

COPECA

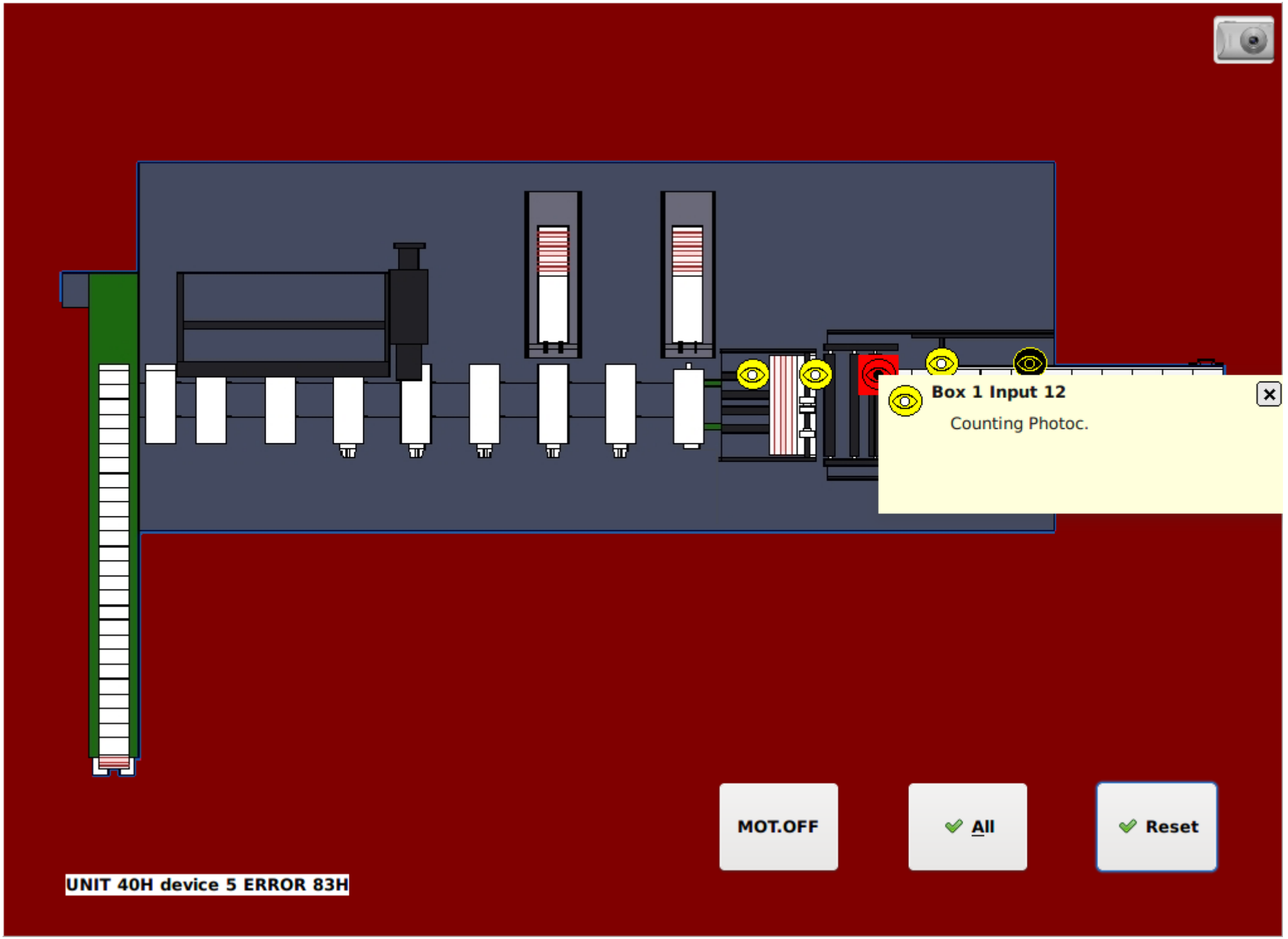


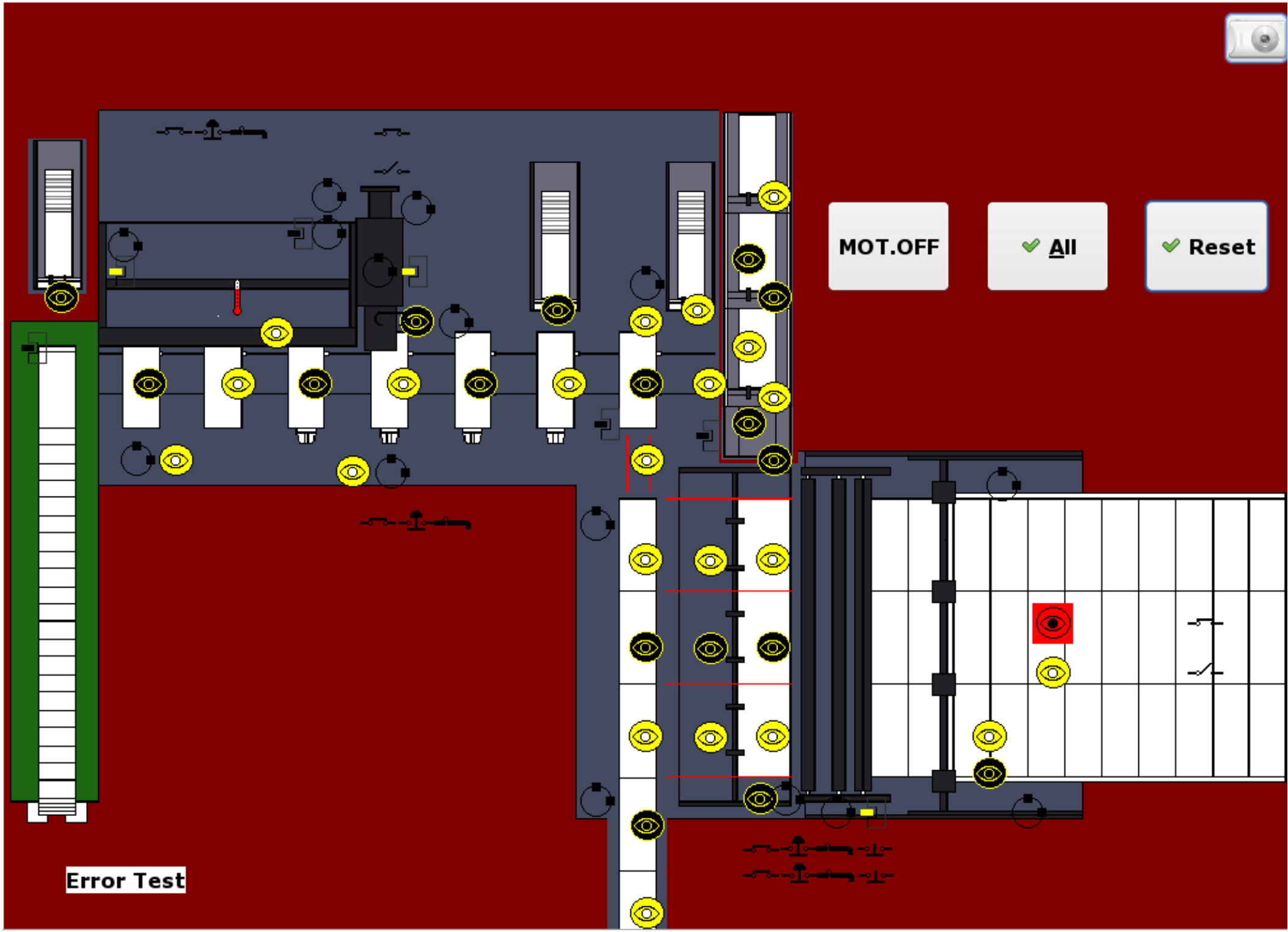
MOT.OFF

All

Reset

UNIT 40H device 5 ERROR 83H


Box 1 Input 12
Counting Photoc.
MOT.OFF
All
Reset
UNIT 40H device 5 ERROR 83H


MOT.OFF
All
Reset
Error Test

Results

- First machine – rather simple – upgrade of previous one – slave units already debugged
- HI: > 20,000 lines Object Pascal – rewritten
- Real-time: ≃ 10,000 lines C - rewritten
- Debugged and tested in simulation
- Install, test and fix details on the real machine: 2 days
- Factory tests – several weeks – passed
- Shipped

Acknowledgements

- Robert Kahn (Intel corp)
- Paolo Mantegazza (Politecnico di Milano) and RTAI team
- Thomas Gleixner and Rt-Linux team
- Florian Klämpfl and FPC team
- Mattias Gärtner and Lazarus team
- The Open Source community
- Aldo Vitrò (NCS computers)