The RFPI management Software layer – library and CLI

Wojciech Tylman

About Me

✓ Wojciech Tylman

D.Sc. In Computer Science, Ph.D. in Electronics

Role

C++ software engineer

Relevant experience

25+ years of experience in C++ programming

Design and implementation of SIL-4 software for rail traffic control and signaling, numerous projects.

Agenda

- The main requirements,
- Specification and scope,
- The functionality and design details,
- Implementation.

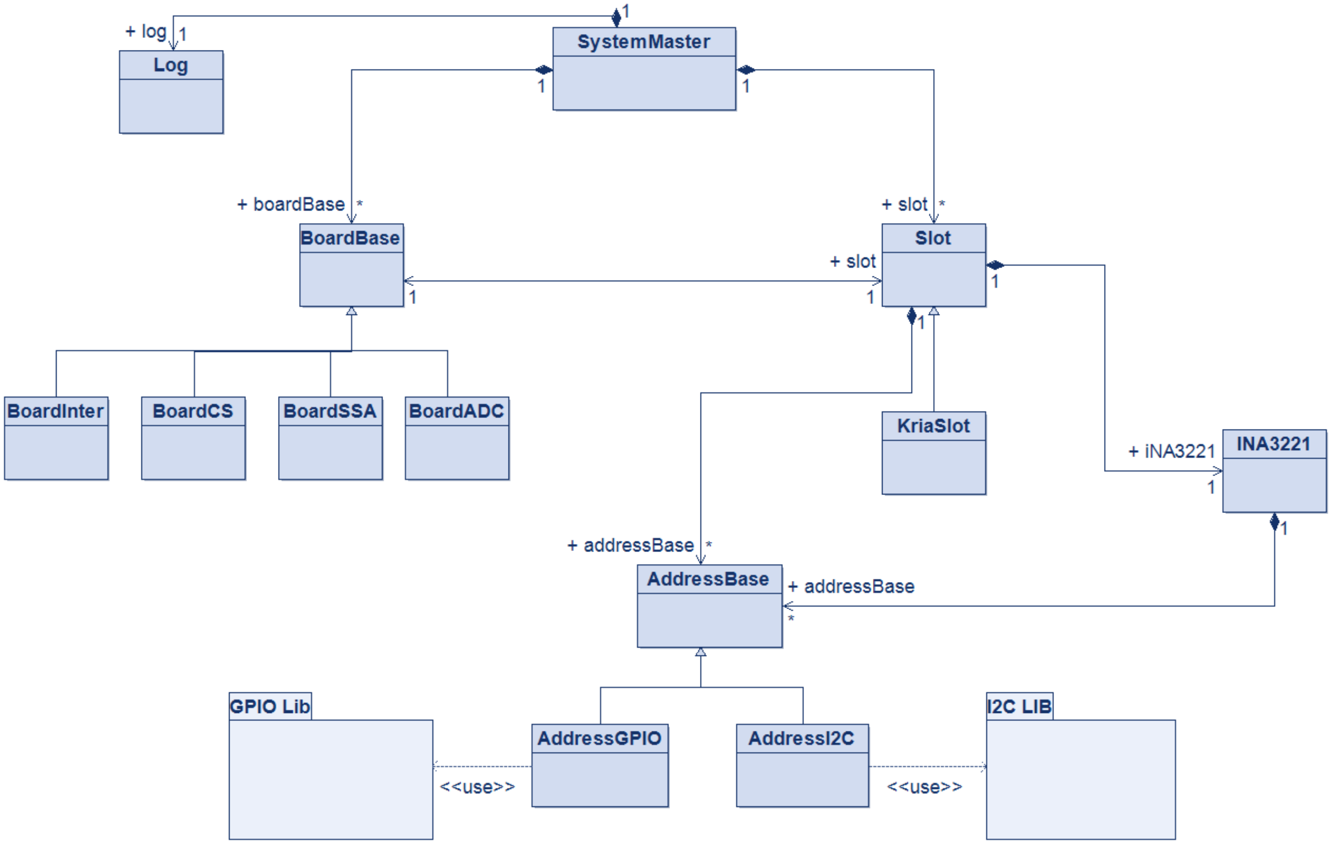
The main requirements

- Software Layer acts as a middle layer between the firmware/hardware and IOC
  - Is the only way that IOC may interact with the system
- Data interaction through I2C and GPIO in required
- Logic for control/monitoring of the carrier must be implemented, e.g., startup/shutdown sequences, overcurrent reporting
- Flexible (i.e., not compiled-in) configuration for various possible hardware setups is advisable
- Command Line Interface (CLI) tool is welcome, to facilitate development and testing

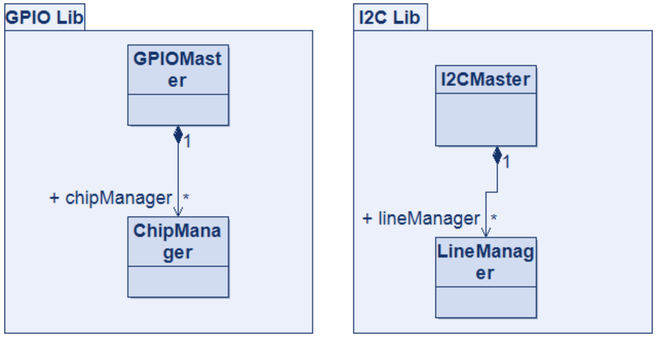
Specification and scope

- Targets the current design of the carrier and extension boards
- Structure allowing easy expansion of the software should alteration to the carrier or new extension boards appear
- Compile-time specification of the carrier configuration, run-time specification of the FMC slot—extension board pairing, GPIO/I2C addresses
- Non-blocking, multithreaded operation for fault monitoring

Design details – class structure



Design details – supporting sub-libraries



Design details – configuration file

```
"top1":{
    "temperature": {
        "line" : 5,
        "slaveAddress": 40
        },
        "relay":{
            "set":{
                "chipNo": 2,
                "pinNo": 25,
                "activeLow" : true
            },
            "get":{
                "chipNo": 2,
                "pinNo": 24
            }
        },
        "presence": {
            "chipNo" : 2,
            "pinNo": 27
        },
```

```
"ina3221": {
    "address": {
        "line" : 5,
        "slaveAddress": 65
        },
        "shunt": [0.01, 0.01, 0.01],
        "avg":  10,
        "vtime":  3,
        "itime":  7
    },
    "status": {
        "line" : 1,
        "slaveAddress": 17
    },
    "power_alert":{
        "chipNo":2,
        "pinNo":26
    },
    "eeprom":{
        "line" : 5,
        "slaveAddress": 80
    }
}
```

Design details – CLI commands

```
Commands common for all boards:
  -r --read           read register/pin
    i:addr:count or i:addr:reg:count for I2C
    g:chip:pin for GPIO
  -w --write          write register/pin
    i:addr:"val1 val2 valn" or
    i:addr:reg:"val1 val2 valn" for I2C,
    g:chip:pin:val for GPIO
  -m --monitor        monitor GPIO pin for events
    chip:pin:events
  --verify            verify hardware configuration
  -p --power          power up/down the system
  -v --voltage        read board supply voltage
    chn (0-3) to select readout channel
  -c --current        read board supply current
    chn (0-3) to select readout channel
  -t --temperature    read board temperature
  -s --status         check board status
  --presence          check board presence
  --typeok            check board type
  --voltageok         check voltage OK for board
  --currentok         check current OK for board
  --healthy           check board complete health
  --currentw          set current warning for slot
    chn:level
  --currentc          set current critical for slot
    chn:level
  --voltagel          set voltage low for slot
  --voltageh          set voltage high for slot
  --powermon          monitor power alerts
  --power             get/set power state
  --fan               get/set fan speed
  --vadj              get/set vadj global signal

  --version           display library version
  -m --man            display manual
```

Implementation – libraries, tools

- g++ cross compilation – PC/x86 host, ARM target
- POSIX Threads (pthreads) for multithreading
- libgpiod for GPIO access