目录

[1 引言 2](#_Toc4316049)

[1.1 编写目的 2](#_Toc4316050)

[1.2 预期读者 2](#_Toc4316051)

[2 设计概述 2](#_Toc4316052)

[2.1 开发环境 2](#_Toc4316053)

[2.2 设计原则和设计要求 3](#_Toc4316054)

[3 系统逻辑设计 3](#_Toc4316055)

[3.1 系统总体结构 3](#_Toc4316056)

[3.2 各模块内部结构 4](#_Toc4316057)

[4 详细设计 5](#_Toc4316058)

[4.1.1 嵌入式Linux编译、配置 5](#_Toc4316059)

[4.1.2 数据库安装配置 6](#_Toc4316060)

[4.1.3 数据表创建 6](#_Toc4316061)

[4.2 SNMP 代理模块设计 7](#_Toc4316062)

[4.2.1 基础库安装配置 7](#_Toc4316063)

[4.2.2 时统MIB文件定义 8](#_Toc4316064)

[4.2.3 数据访问模板代码生成 10](#_Toc4316065)

[4.2.4 数据访问模板代码修改说明 11](#_Toc4316066)

[4.3 子板交互模块设计 12](#_Toc4316067)

[4.3.1 子板通信接口 12](#_Toc4316068)

[4.3.2 上报数据存储进程 12](#_Toc4316069)

[4.3.3 操控命令处理进程 12](#_Toc4316070)

[4.3.4 报警灯控制进程 12](#_Toc4316071)

[4.4 时间同步模块 12](#_Toc4316072)

[4.5 软件升级交互模块设计 13](#_Toc4316073)

[4.5.1 上位机通信协议 13](#_Toc4316074)

[4.5.2 子板通信协议 13](#_Toc4316075)

[4.5.3 软件升级处理 13](#_Toc4316076)

[4.6 QT设备操控界面设计 13](#_Toc4316077)

[4.7 操控上位机设计 13](#_Toc4316078)

[附录 13](#_Toc4316079)

[参考资料 13](#_Toc4316080)

# 引言

本文档简要描述了监控板软件的各个模块的功能及其相互关系

## 编写目的

本文档基于《软件需求分析》文档编写。监控板软件负责监测设备的运行状况、修改设备的运行参数。监控板软件支持SNMP协议的网络管理接口。另外，本文档也描述了与监控板软件配合的PC端上位机软件的设计，上位机软件功能与监控板上运行的操控软件功能相同。

## 预期读者

* 用户；
* 开发人员；
* 项目经理；
* 营销人员；
* 测试人员；
* 文档编写人员；

# 设计概述

本节描述现有开发条件和需要实现的目标，说明进行软件设计时应该遵循的设计原则和必须采用的设计方法。

## 开发环境

* SNMP Agent模块

操作系统：Unbuntu14.04+Windows 7

开发工具：Gcc、Eclipse、Visual Studio 2010

开发支持环境：net-snmp-5.7.3

* Qt操控界面模块

操作系统：Unbuntu14.04 + Windows 7

开发工具：Gcc、Eclipse、Qt Creator 4.0.1、Visual Studio 2010

开发支持环境：qt-everywhere-opensource-src-5.6.1

添加软件升级功能，

* 子板交互

操作系统：Unbuntu14.04+Windows 7

开发工具：Gcc、Visual Studio 2010、CANTest

开发支持环境：Socket-CAN

* 操控上位机软件

操作系统：Unbuntu14.04+Windows 7

开发工具：Gcc、Visual Studio 2010

开发支持环境：net-snmp-5.7.3、qt-everywhere-opensource-src-5.6.1

## 设计原则和设计要求

* 界面简洁、操作方便；
* 模块独立，系统可维护性高；
* 可靠性高；

# 系统逻辑设计

本节内容主要根据软件产品需求规格说明书建立系统的逻辑模型。此种模型暂时与系统的物理因素(例如：计算机、数据库管理系统)无关。它是系统需求与物理实现的中间结构，它的主要结果是建立：系统结构图、系统界面结构图、系统出错处理、以及系统开发技术说明。

## 系统总体结构



‑系统结构框图

监控板软件由三大模块组成：SNMP Agent、子板交互模块、Qt操控界面模块。SNMP Agent通过SNMP协议设备操控上位机交互，子板交互模块与各功能板之间通过CAN协议交互。Qt操控界面模块完成操控界面

注：后续添加软件升级模块，软件升级模块完成系统所有软件的升级功能。

## 系统工作HH机制



## 各模块内部结构

* SNMP Agent模块内部逻辑结构图



* 子板交互模块内部逻辑交互图



与各子板采用CAN总线通信分层结构如下：



* Qt操控界面模块内部逻辑结构图



# 详细设计

## 嵌入式Linux编译、配置

### yocto SDK编译和使用

参考: <https://www.yoctoproject.org/docs/2.6.1/sdk-manual.html>

## 数据库设计

### 数据库安装配置

* 下载sqlite3：<https://www.sqlite.org/src/tarball/sqlite.tar.gz>;
* ubuntu平台sqlite数据库测试环境安装：

tar xzvf sqlite.tar.gz

cd sqlite

./configure

make

make install

* ARM平台sqlite3数据库基础库编译

tar xzvf sqlite.tar.gz

cd sqlite

CC=arm-poky-linux-gnueabi-gcc ./configure --prefix=/work/tmp/ --host=arm-linux

make

make install

### 数据库组成

每个槽位板卡的数据存放在一个表中，一共有26个板卡，所有有26个板卡数据表

### 板卡数据表



每个槽位根据插入的板卡类型确定数据表的格式，一共有以下10种格式的板卡数据表：

* 电源板上报数据表：

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡出错标志：0正常；1故障 |
| period | integer | 数据询问周期 |
| current | integer | 电流大小 |

* 接收机板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
| state | integer | 运行状态：0未锁定；1锁定 |
| antenna | integer | 天线故障标志：0正常；故障 |
| satellite | integer | 收星数 |
| latitude | real | 经度 |
| longitude | real | 纬度 |
| height | integer | 高度（？单位是米） |
| freq | integer | 频点 |
| loc\_mode | integer | 定位模式 |

* 外频板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
| state | integer | 运行状态：0未锁定；1锁定 |

* 时频板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
| ref | integer | 参考源板卡地址 |
| state | integer | 状态：0跟踪；1同步中；2同步完成；3保持；4故障 |
| time\_diff | integer | 时差 |
| freq\_diff | integer | 频差 |
| syn\_mode | integer | 同步模式：0手动；1自动 |

* 切换板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
| sw\_mode | integer | 切换模式 |
| TF\_sel | integer | 时频输出选择 |

* 分配板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
|  |  |  |

* NTP板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
|  |  |  |

* PTP板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
|  |  |  |

* B码板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
| mr | integer | 调制比(modulation ratio) |
| amp | integer | 输出幅度 |
| output\_state | integer | 用低8位表示8路B码输出状态，比如最低位为0，表示第1路输出正常，最低为为1，表示第1路输出故障 |

* 10M板上报数据表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| flag | integer | 板卡类型 |
| period | integer | 数据询问周期 |
| run\_state | integer | 0捕获；1跟踪；2锁定；失锁 |
| output\_state | integer | 用低4位表示4路10M输出状态 |

### 操控命令表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| ID | integer | 索引号，定义为主键 |
| timestamp | integer | 时间戳，从1970年1月1日开始的秒数 |
| nID | integer | 板卡地址 |
| boardtype | integer | 板卡类型 |
| cmdtype | integer | 操控命令码 |
| cmdparam | blob | 操控参数 |

### 板卡在位信息表

|  |  |  |
| --- | --- | --- |
| 列名 | 类型 | 描述 |
| boardtype | integer | 板卡类型，行号与槽位号对应 |
| flag | integer | 板卡状态 |

注：后续改进，为了减少flash写入次数，增加临时数据库和数据库管理模块，数据交互在临时数据库中完成，数据库管理模块负责定期保存临时数据库到flash;

采用动态表：在系统运行过程中，根据各槽位插入的板卡，创建表，根据板卡种类，确定表的列。

## SNMP 代理模块设计

### 基础库安装配置

* 下载net-snmp-5.7.3.tar.gz：<http://www.net-snmp.org/download/>;
* windows平台下net-snmp开发测试环境安装：下载运行net-snmp-5.6.1.1-1.win32.exe
* ubuntu平台net-snmp开发测试环境安装：

tar xzvf net-snmp-5.7.3.tar.gz

cd net-snmp-5.7.3

./configure

make

make install

* ARM平台net-snmp基础库编译

tar xzvf net-snmp-5.7.3.tar.gz

cd net-snmp-5.7.3

./configure --prefix=/work/snmp\_arm --disable-shared --disable-scripts --disable-ipv6 --disable-manuals --disable-embedded-perl --host=arm-linux --target=arm-linux --build=i686-linux --with-cc=arm-none-linux-gnueabi-gcc --with-ar=arm-none-linux-gnueabi-ar --enable-mini-agent --enable-debugging --with-endianness=little --with-openssl=internal --enable-as-needed --disable-ucd-snmp-compatibility

make

make install

### 时统MIB文件定义

具体定义见文件FTS-MIB.txt

* 数据上报节点定义：

upload\_data OBJECT IDENTIFIER ::= {ftsGlobalModule 1}

data\_pwr\_table OBJECT-TYPE

SYNTAX SEQUENCE OF Data\_pwr\_entry

MAX-ACCESS read-write

::= {upload\_data 1}

data\_pwr\_entry OBJECT-TYPE

SYNTAX Data\_pwr\_entry

MAX-ACCESS read-write

INDEX {id INTEGER}

::= {data\_pwr\_table 1}

Data\_pwr\_entry ::=

SEQUENCE {

id

INTEGER,

timestamp

INTEGER,

flag

INTEGER,

period

INTEGER,

current\_size

INTEGER

}

id OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {data\_pwr\_entry 1}

timestamp OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {data\_pwr\_entry 2}

nID OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {data\_pwr\_entry 3}

flag OBJECT-TYPE

period OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {data\_pwr\_entry 5}

current\_size OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {data\_pwr\_entry 6}

* 控制命令节点定义：

ctl\_cmd OBJECT IDENTIFIER ::= {ftsGlobalModule 2}

pwr\_period OBJECT-TYPE

SYNTAX OCTET STRING

MAX-ACCESS write-only

DESCRIPTION

"nID period"

::= {ctl\_cmd 1}

b\_period OBJECT-TYPE

SYNTAX OCTET STRING

MAX-ACCESS write-only

DESCRIPTION

"nID period"

::= {ctl\_cmd 901}

b\_modulate\_index OBJECT-TYPE

SYNTAX OCTET STRING

MAX-ACCESS write-only

DESCRIPTION

"nID modulate index"

::= {ctl\_cmd 902}

b\_amp OBJECT-TYPE

SYNTAX OCTET STRING

MAX-ACCESS write-only

DESCRIPTION

"nID amplitude"

::= {ctl\_cmd 903}

* 系统状态节点定义：

sys\_info OBJECT IDENTIFIER ::= {ftsGlobalModule 3}

slot\_info\_table OBJECT-TYPE

SYNTAX SEQUENCE OF Slot\_info

MAX-ACCESS read-only

::= {sys\_info 1}

slot\_info\_entry OBJECT-TYPE

SYNTAX Slot\_info\_entry

MAX-ACCESS read-only

INDEX {id INTEGER}

::= {slot\_info\_table 1}

Slot\_info\_entry ::=

SEQUENCE {

id

INTEGER,

boardtype

INTEGER

}

id OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {slot\_info\_entry 1}

boardtype OBJECT-TYPE

SYNTAX INTEGER

MAX-ACCESS read-only

::= {slot\_info\_entry 2}

### 数据访问模板代码生成

* 电源板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_pwr\_table
* 接收机板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_gnss\_table
* 外频板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_outref\_table
* 时频板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_tf\_table
* 切换板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_sw\_table
* 分配板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_alloc\_table
* NTP板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_ntp\_table
* PTP板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_ptp\_table
* B码板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_b\_table
* 10M板上报数据表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: data\_10m\_table
* 板卡在位信息表：MIBS=+ FTS-MIB mib2c -c mib2c.iterate.conf FTS-MIB:: slot\_info\_table
* 控制命令： MIBS=+ FTS-MIB mib2c -c mib2c.scalar.conf FTS-MIB:: pwr\_period

注：完成以上操作后，把代码合成到一个文件中。

### 数据访问模板代码修改说明

MIB库中的变量由两种类型，一种是只读的table类变量，另一种是只写的scalar类变量

table类变量包括

* table类变量获取：

int xxxTable\_handler(...)

{

...

switch (reqinfo->mode) {

case MODE\_GET:

for (request=requests; request; request=request->next) {

...

switch (table\_info->colnum) {

case COLUMN\_xxx:

//添加数据库读取代码

break;

......

}

* scalar变量设置：

int handle\_xxx(...)

{

......

switch(reqinfo->mode) {

......

case MODE\_SET\_ACTION:

//添加数据库写入代码

......

}

## 子板交互模块设计

### 监测数据接收与存储



### 操控命令处理进程



### 报警灯控制进程



### 时间同步进程



### CAN通信接口

* 发送数据包：int can\_send\_packet(int nID, int protocal, char \*buf, int len);
* 接收数据包：int can\_recv\_packet(int nID, int protocal, char \*buf, int \*len);

## 软件升级交互模块设计

### 上位机通信协议

### 子板通信协议

### 软件升级处理

## QT设备操控界面设计

## 操控上位机设计

# 附录

## 参考资料

* RFC 1213, Management Information Base for Network Management of TCP/IP-based internets: MIB-II；
* RFC 2578, Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)；
* RFC 1905, Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)；
* Net-SNMP Wiki, http://www.net-snmp.org/wiki；
* Net-SNMP FAQ, http://www.net-snmp/wiki/index.php/FAQ；
* Essential SNMP, 2nd Edition, By Douglas Mauro, Kevin Schmidt；
* Managing Internetworks with SNMP, By Mark A. Miller；
* Qt图形界面编程入门，仇国巍著，2018年
* Qt 5.2.1 Reference Documentation；
* C++ GUI Programming with Qt 4, Second Edition by Jasmin Blanchette; Mark Summerfield；
* 深入理解Net-SNMP ，张春强著；
* CAN Specification Version 2.0, http://www.semiconductors.bosch.de/pdf/can2spec.pdf；
* SocketCAN - Controller Area Network, Linux Kernel Documentation/networking/can.txt.