On-Board Software Design for HumSAT Demonstrator

some insights

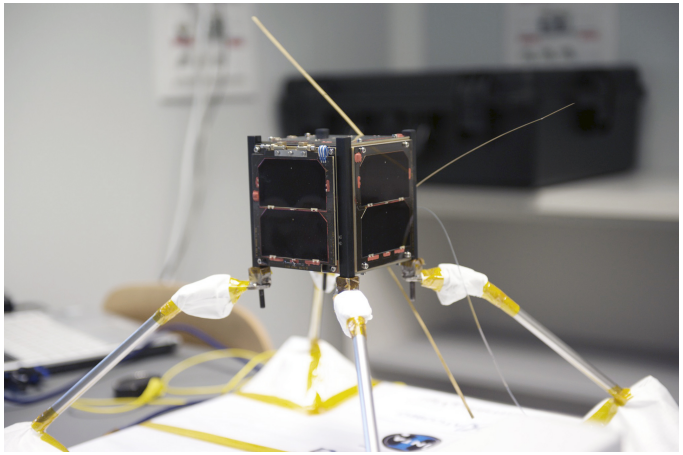
Prof.Dr. Arno Formella

Computer Science Department
Laboratorio de Informática Aplicada (LIA)
University of Vigo

September 3rd, 2013

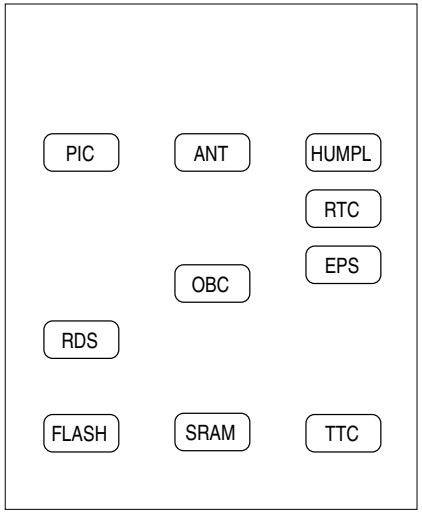
The Hardware

I'm cheating, it's XaTcobeo but looks the same :-)



HumSAT Demonstrator

(spacecraft) system overview (software view)



- Electric Power System
- On-Board Computer
- Tracking, TeleCommand
- FLASH memory
- Static Random Access Memory
- Real Time Clock
- Radio Dose Sensor
- Power Integrated Control
- HumSAT PayLoad
- Antenna

Software requirement specification

overview

- general requirements
- functional requirements
- performance requirements
- interface requirements
- operational requirements
- resource requirements
- design requirements
- security/privacy requirements
- other requirements

Software requirement specification
some examples

General requirements:

- must run as autonomous application whenever initiated
- must run on onboard hardware
- must manage all payloads
- etc.

Software requirement specification
some examples

Functional requirements:

- must wait half an hour before activating radio equipment
- must deploy antenna
- must collect housekeeping data of payloads and subsystems
- must execute a task scheduler with programmed operations
- etc.

Software requirement specification
some examples

Interface requirements:

- must interact with TTC via IIC-bus
- must interact with EPS/RTC/HUMPL via IIC-bus
- must interact with RDS via ADC interface
- etc.

Software requirement specification
some examples

Operational requirements:

- must operate operational mode state machine
- must reset watchdog periodically
- must powerdown periodically before hardware powerdown
- etc.

Software requirement specification

some examples

Resource requirements:

- the software (and its data) must fit into available SRAM memory
- the bootloader must fit into available BRAM memory
- etc.

Software requirement specification

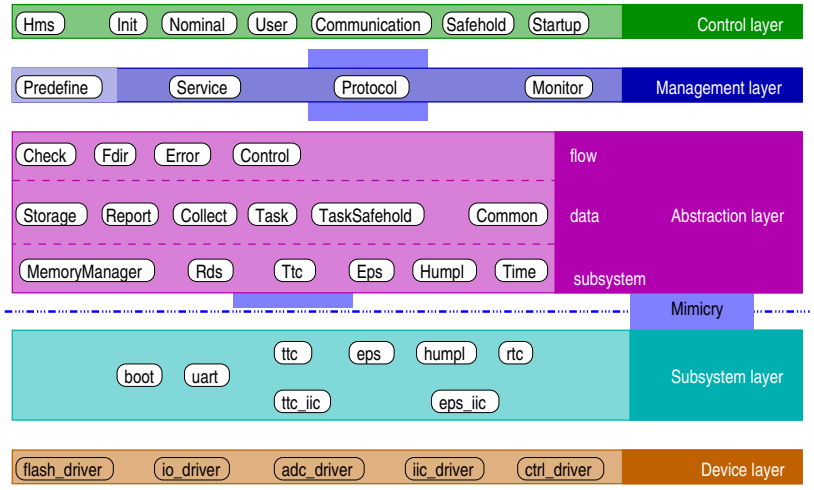
some examples

Design requirements:

- must be written in C
- must be compilable/developed in the EDK development kit of Xilinx
- must be modular to allow for integration and test activities
- etc.

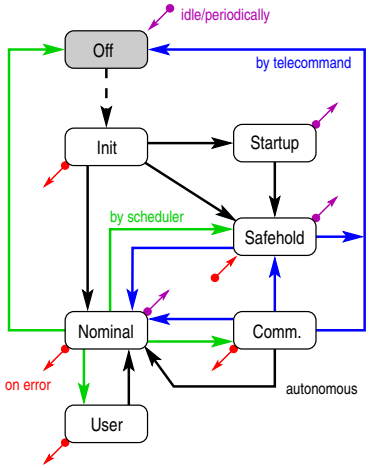
Software design

layered structure



Operational modes

overview



- upmost level is a simple state machine
- 6 principal modes
- different type of mode transitions
  - autonomous by program flow
  - by telecommand executing request
  - by scheduler executing task
  - on error due to failure condition
  - periodic or idle powerdown

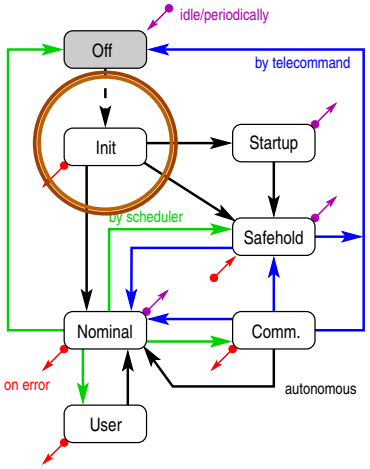
Our main loop

mode change state machine

```
1  main() {
2    Reset();
3    Init();
4    while(1) {
5      switch(mode) {
6        case STARTUP: Startup(); break;
7        case NOMINAL: Nominal(); break;
8        case COMMUNICATION: Communication(); break;
9        case USER: User(); break;
10       default: Error();
11       case SAFEHOLD: Safehold(); break;
12     }
13   }
14   for(;;);
15 }
```


Operational modes

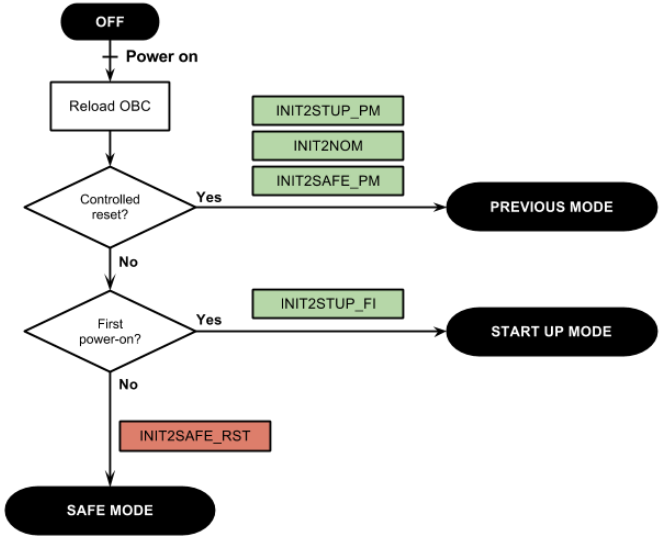
init mode



- bootstrap
- setting of system time
- initialization of data structures
- health check
- mode selection

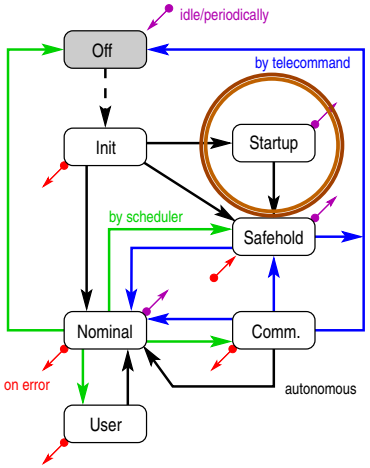
Operational modes

init mode



Operational modes

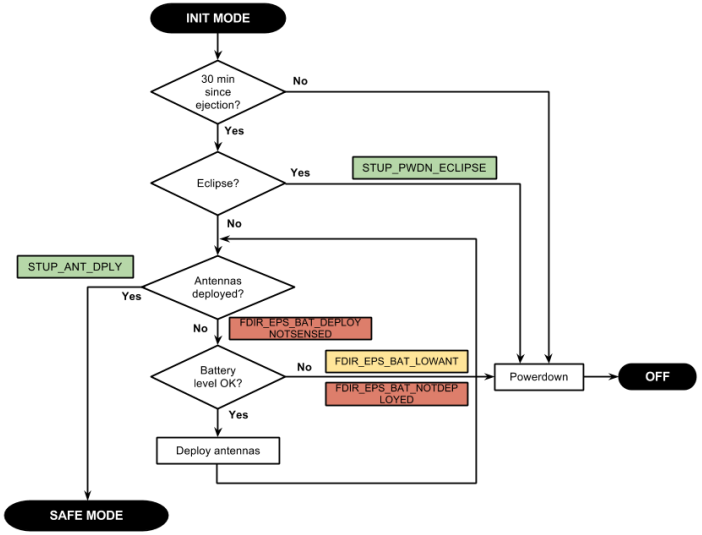
startup mode



- wait half an hour before any activity
- deploy antenna
- change mode to safehold mode

Operational modes

startup mode



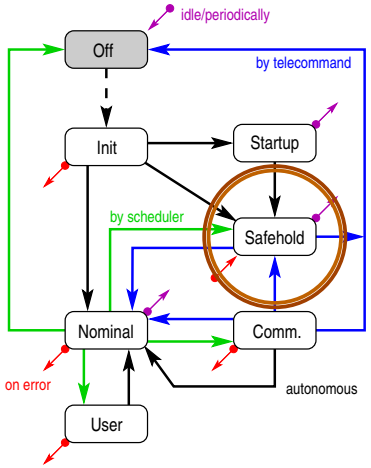
Startup mode

mode change state machine

```
1  void Startup() {
2    if (!MemManAntennaIsUnDeployed()) {
3      MemManMarkFirstHalfHourPassed();
4      MemManMarkFirstAntennaDeployDone();
5      MemManMarkAntennaDeployed();
6    }
7    else if (!ControlAntennaIsDeployed()) {
8      if( /* WithinFirstHalfHour */ )) {
9        StartupWaitFirstHalfHour(); // may powerdown
10     }
11     ControlHandleEclipse();        // may powerdown
12     StartupDeployAntenna();
13   }
14   ControlModeChange(MODE_SAFEHOLD);
15 }
```


Operational modes

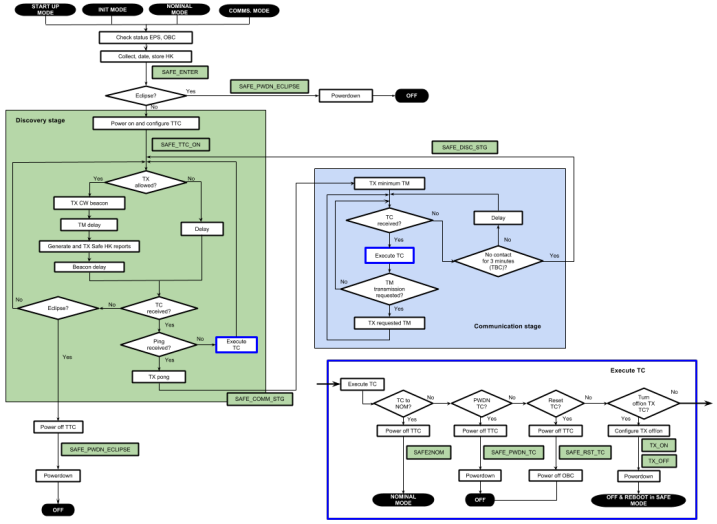
safehold mode



- switch-off any payload
- initialize safehold tasks
- check eclipse condition
- switch-on and configure TTC
- run safehold task scheduler loop
- handle no-transmission behavior

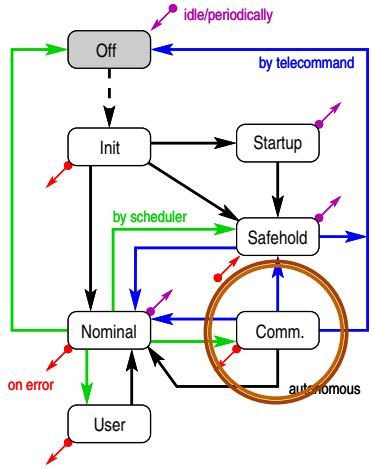
Operational modes

safehold mode



Operational modes

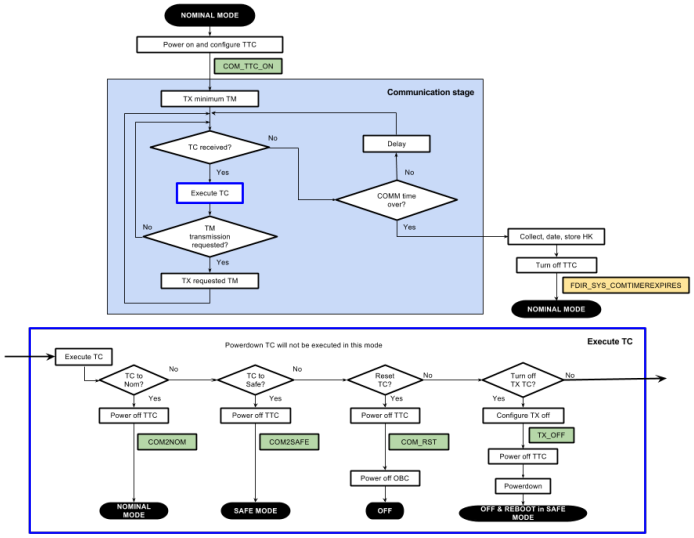
communication mode



- perform health checks
- alternate between
  - receiving of telecommand
  - sending of telemetry
- for a certain amount of time

Operational modes

communication mode



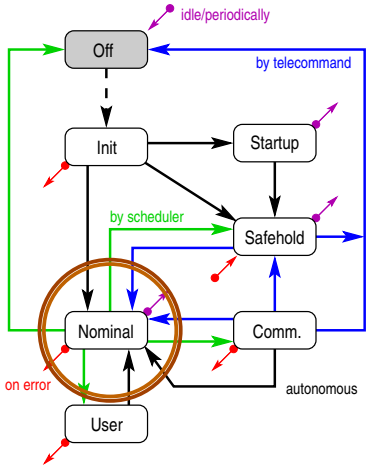
Communication mode

mode change state machine

```
1 void Communication() {
2   TtcOnAndConfigure();
3   CheckPointCommunication001();
4   CommunicationStage();
5     // wait for the TTC to finish transmitting the ACK
6   timer_wait(TIME_WAIT_FOR_TTC_TO_SEND_ACK_US);
7   TtcOff();
8   CheckPointCommunication34567();
9 }
```


Operational modes

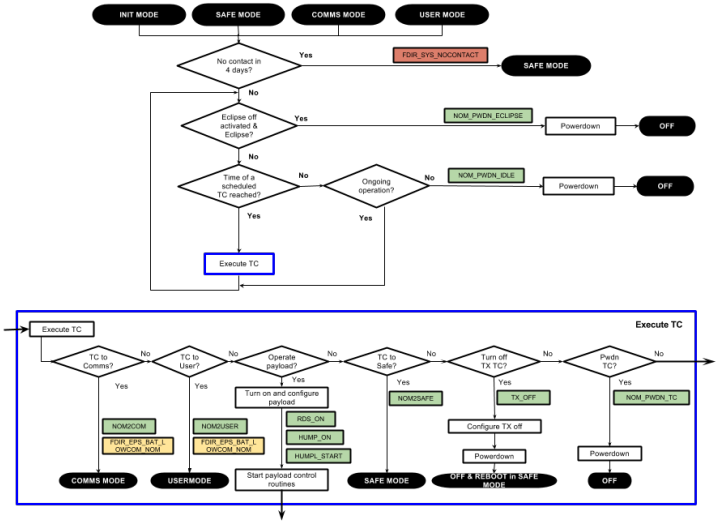
nominal mode



- active HumSAT payload if required
- execute programmed service data unit request
- operate payloads as scheduled
- collect housekeeping data

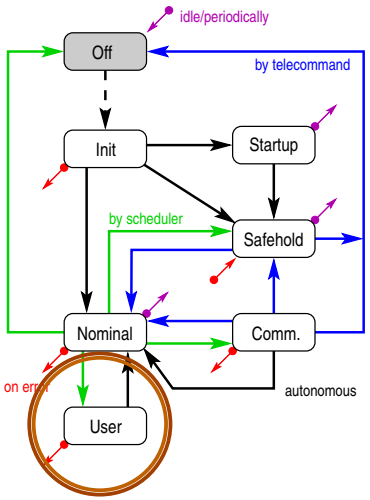
Operational modes

nominal mode



Operational modes

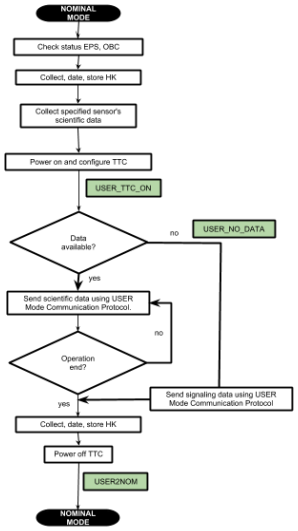
user mode



- send user data for specified user filter in a loop

Operational modes

user mode

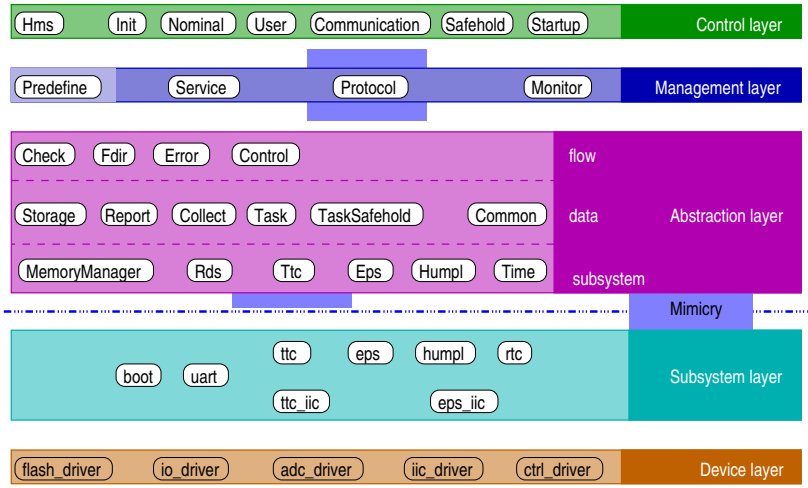


User mode

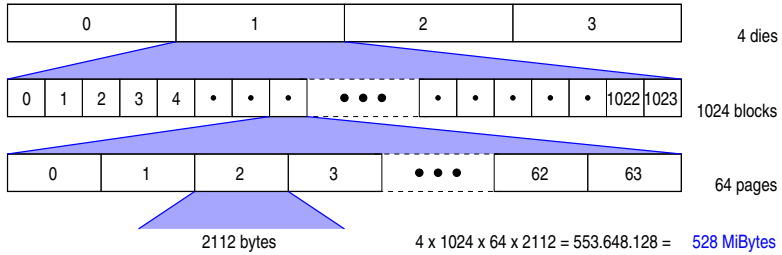
mode change state machine

```
1  void User() {
2    UserInit();  // reads references for user mask
3    TtcOnAndConfigure();
4    /* Set user time window */
5    do {
6      ControlHandleWatchdog();
7      if (!number_of_references) {
8        UserSendSignaling();
9        continue;
10     }
11     RingReadSdu(...); // reads user data cyclic
12     UserSend(&sduRpBox.sdu_rp_humpl_event);
13   } while (!UserIsTimeout());
14   TtcOff();
15   ControlModeChange(MODE_NOMINAL);
16 }
```

Software design
layered structure



FLASH memory
dies, blocks, pages, bytes ...



- NAND gate based flash technology
- can erase only blocks (put all to ones)
- can write bytes sequences that don't cross page boundaries
- 4096 blocks as base units

Storage of data in FLASH memory
what is stored?

basically the following information is dynamically stored:

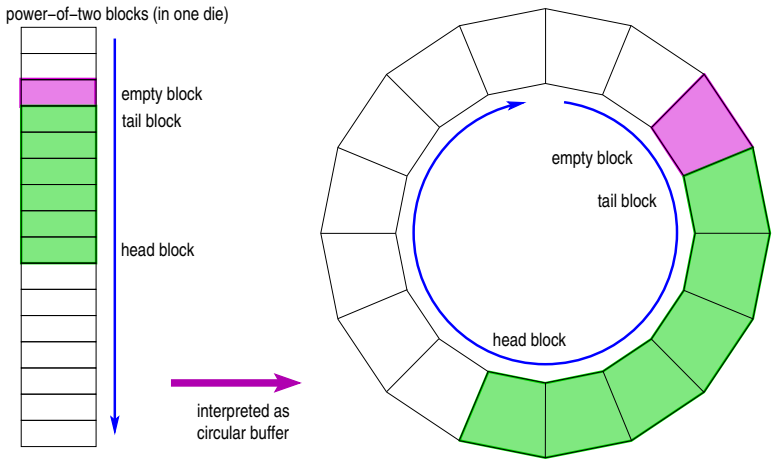
- reports (telemetry) generated for different reasons,
  e.g., telecommands or user data from HumSAT payload
- report and event report definitions
- housekeeping parameter values (periodically, or on demand)
  these values can be monitored
- system state to recover after powerdown
  (includes task schedulers)

Storage of data in FLASH memory
how is it stored?

reports	in a circular buffer
definitions	in modifiable and initializable blocks
parameters	in a set of two-block storage
system state	in double buffers (with state machines)

Circular buffer storage

simple data structure to store telemetry (reports)



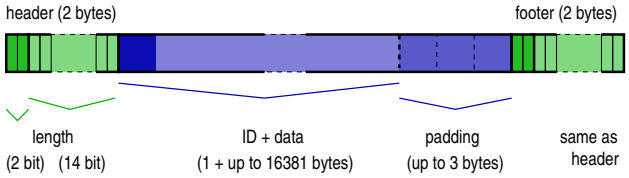
Circular buffer

basic invariants

- power-of-two number of blocks (makes index calculation easy)
- at least one empty block
- occupied blocks are consecutive
- each block has mark at first byte: empty or used
- maintaining:
  - head pointer: points to mark of head block
  - tail pointer: points to mark of tail block
  - top pointer: points to first unused entry in head block
- head/tail/top held in system state

Circular buffer entries

fill circular buffer perfectly



- header and footer are coding length of data+padding field
- empty entry occupies 4 bytes
  (either fits or previous entry uses padding)
- permits forward/backward iterates jumping with header/footer info
- 0xFFFF header (or footer) means unoccupied
- zero ID codes invalidated entry

Circular buffer initialization

is quite simple

- assume initially empty (all 0xFF) die
- initialize:
  - place head and tail to some block
  - mark block as used
  - place top to first entry
- auto re-initialize (head, tail, and/or top got lost):
  - place head to some block
  - search with head blockwise forward til used block found
  - place tail there
  - search with head blockwise forward til empty block found
  - step back one block with head
  - search with tail blockwise backward til empty block found
  - step forward one block with tail
  - search with top for first empty entry in head

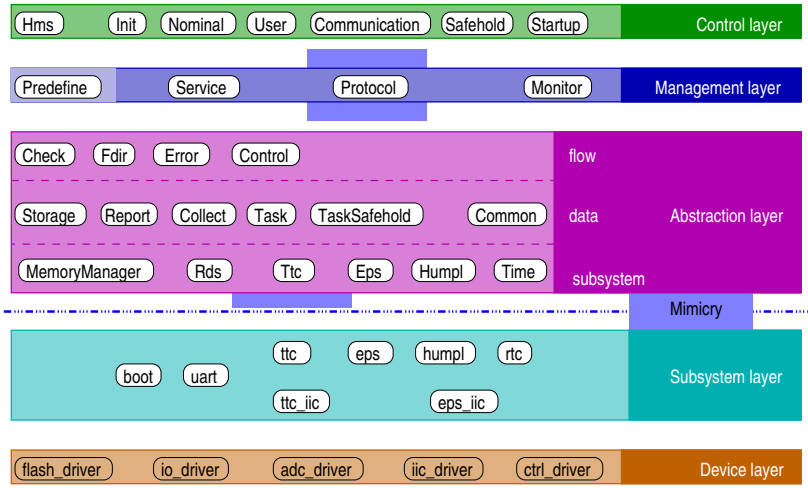
Circular buffer

is quite robust, but still things to do...

- header and footer entries must be equal
- could use SECDED (single error correction double error detection) code on header/footer (e.g., hamming code (16,11,4), allows data size up to 512 bytes)
- could use mark to handle unusable blocks (blocks to ignore)

Software design

layered structure



Storage of report definitions

they say what to gather...

reports are either stored in the circular buffer or transmitted directly to ground (communications and safehold mode)

- event reports
  - generate a report whenever a certain run-time point is reached
  - three types: progress, low-severity and high-severity error
- answers to service data request, i.e., telemetry generated directly by telecommands
- housekeeping and diagnostic data reports (periodically)
- payload data reports (two types)
- monitoring alert reports

Event reports

there definitions and generated reports

- 230 event report definitions
- can be activated/deactivated
- are generated at program points
- bit field for parameters

- event ID
- timestamp of event
- values of included parameters
- size is implicit

Housekeeping and diagnostic reports

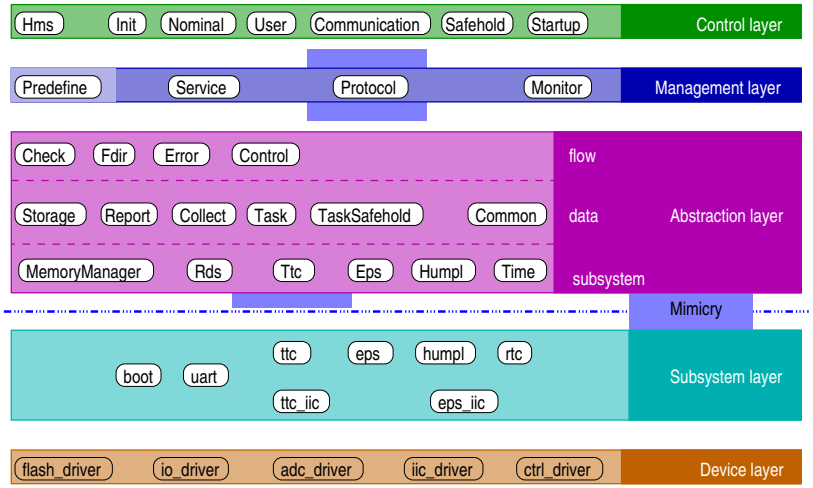
there definitions and generated reports

- 20 diagnostic definitions (safehold task scheduler)
- 19 housekeeping definitions (nominal task scheduler)
- can be activated/deactivated
- are generated every now and then
- bit field for parameters
- parameter values monitorizable

- report ID
- timestamp of collection
- values of parameters
- size is implicit

Software design

layered structure



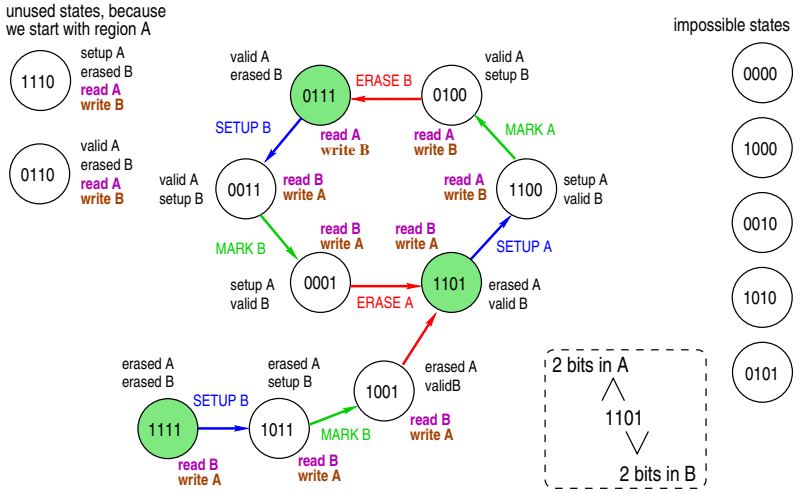
Double buffer state storage

to store system state inbetween powerdowns

- HumSAT powers down whenever possible
- every once and then required by hardware
- unexpected powerdown may occur
- therefore:
  - system state is stored in a two double buffers
  - one for constants modifiable from ground, and
  - one for state variables
  - the buffers are loaded/stored alternatively
  - a four bit (actually bytes) state machine controls the process
- the state machine can live with an erased flash

Double buffer state machine

fallback on unexpected power down



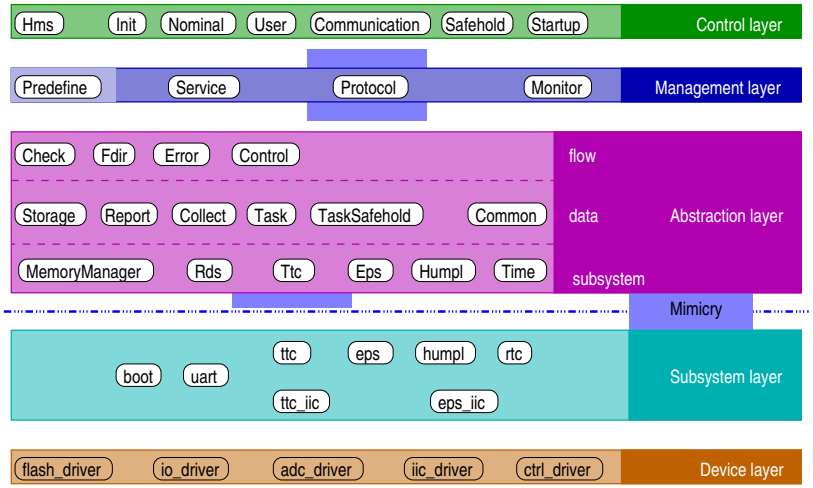
Task scheduling

various task schedulers

- schedulers
  - vital tasks: eclipse check, watchdog reset, periodic powerdown
  - nominal tasks: service data unit requests, periodic operation of payloads, collection (and monitoring) of housekeeping data
  - safehold tasks: switch between discovery and communication stage, collection (and monitoring) of diagnostic data
- priority queue as constant size array with linear search
- priority queue as array embedded sorted linear list

Software design

layered structure



Simulation environment

HumSAT on a laptop... not on-board to test on ground

HDS HumSAT debug service

HGS HumSAT ground simulation

HMS HumSAT mission software

Simulation environment

HumSAT on a laptop... HDS

HumSAT debug service

- emulates UART connection via TCP socket
- currently only output from HumSAT implemented

Simulation environment
HumSAT on a laptop... HMS

HumSAT mission software

- same C-code as flight software
- mimicries logical behavior of subsystems
- included are: FLASH, EPS, TTC, RTC, RDS, HUMPL, UART, watchdog, powerdown
- still simple debug emulation, but extensible to more sophisticated emulator (thought for operation training equipment)
- connectable to any host with/without communication protocol baesd on TCP or UDP protocol

Simulation environment
HumSAT on a laptop... HGS

HumSAT ground simulation

- emulates either RX or TX ground station via UDP sockets
- TX ground station with command line interface for telecommands
- RX ground station with report and beacon interpreter
- simulation on service data units (requests/reports), or
- with full communication protocol,
- hence connectable to real ground station (or what-so-ever)

Simulation environment
HumSAT on a laptop... HGS, how it looks like

```
 HumSAT ground simulation software

hgs: HumSAT epoch Sun Jan  1 00:00:00 2012
hgs: epoch offset 52796523.000000 seconds
hgs: starting humsat ground simulation software
hgs: sending to hms at localhost
hgs: sizeof xtc_frame_t: 40

hgs: use ?? for help

hgs>
```

Simulation environment
HumSAT on a laptop... HGS, how it looks like

```
hgs> ??

  ad(d)          an(tenna)      ch(eck)        cl(ear)
  co(py)         cr(itical)     def(ine)       del(ete)
  du(mp)         ec(lipse)      ep(s)          er(ase)
  ev(ent)        g(et)          he(ater)       hu(mpl)
  i(nterval)     l(oad)         mod(e)         mon(itor)
  n(ame)         pa(rameter)    pi(ng)         po(werdown)
  pr(otocol)     ra(te)         rd(s)          rep(ort)
  rese(t)        rest(ore)      se(t)          shi(ft)
  sho(w)         ta(sk)         te(lemetry)    ti(me)
  tr(ansmit)     tt(c)          u(nknown)      w(rite)
```

Simulation environment

HumSAT on a laptop... HGS, how it looks like

```
hgs: use  ?command  for more help on command

    !q(uit)                quit

    !c(lear)               clear history
    !e(xecute) filename    execute script/history file
    !r(ead) filename       read history file
    !s(et) wait_time       set script wait time
    !w(rite) filename      write history file

hgs>
```

Simulation environment

HumSAT on a laptop... HGS, how it looks like

```
hgs> ?mon
use: mon(itor) c(lear)|l(ist)|r(eport)
     mon(itor) d(elete) <id>:1

hgs> ?pa
use: pa(rameter) <id>:1 <interval>:2
                 <rep_limit>:1 <rep_delta>:1
       [ l(imit) <low>:2 <rid>:1 <high>:2 <rid>:1 ]
       [ d(elta) <low>:2 <rid>:1 <high>:2 <rid>:1 ]
       [ e(xpected) <value>:2 <rid>:1 ]

hgs> ?ad
use: ad(d) <when>:4 r(elative)|a(bsolute) command
```

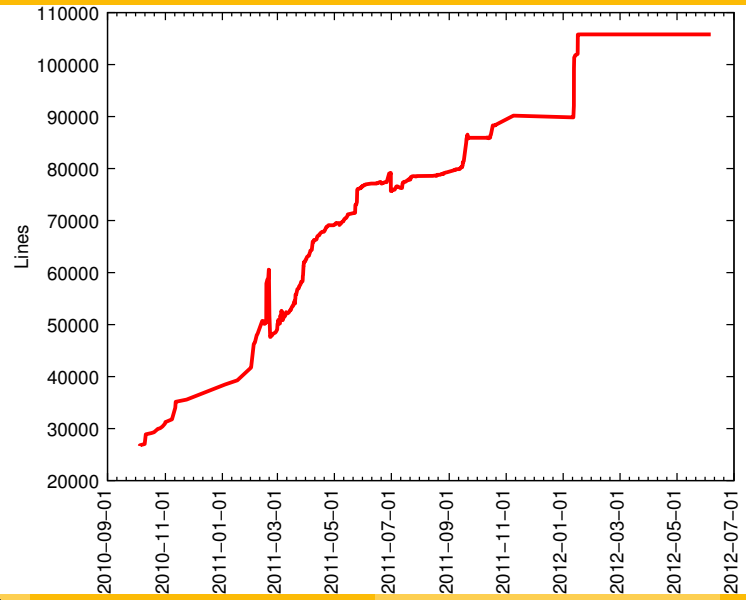
Software management

where's the code...

- version control system git
- embedded documentation with doxygen and LaTeX

HumSAT software evolution

XaTcobeo lines of code



HumSAT software evolution

HumSAT demonstrator lines of code