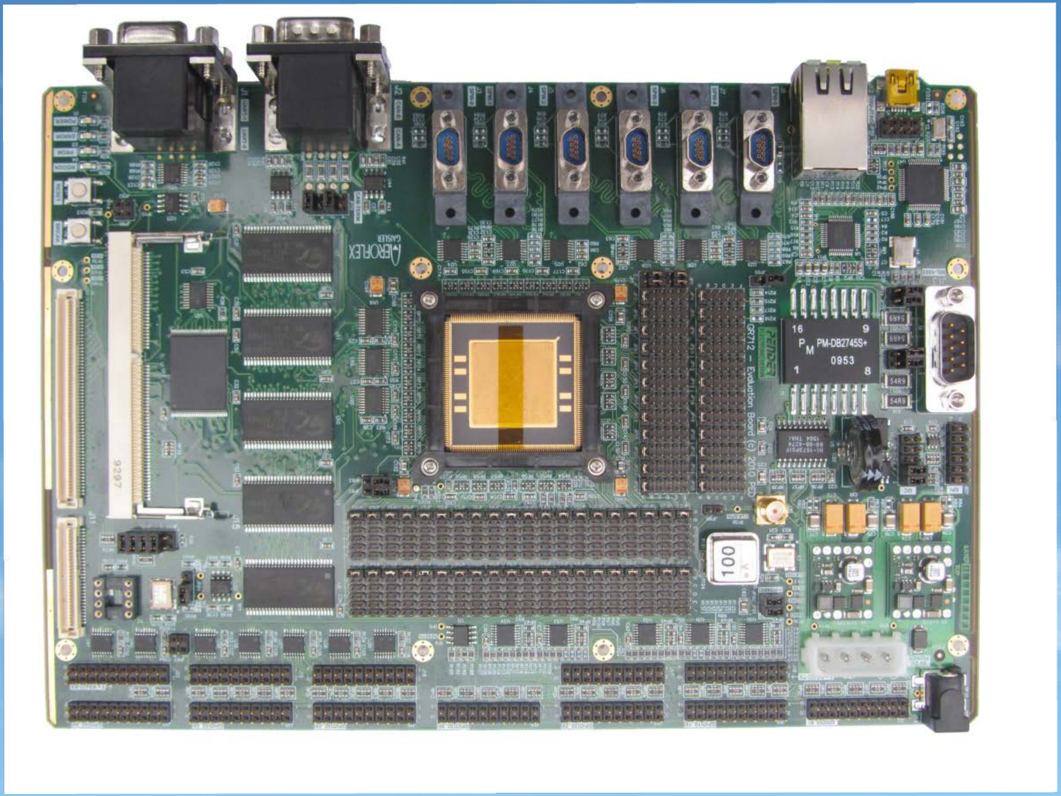
L3Obis: Leon3-based Onboard Instrument Software
A case study using TASTE



Daniel Tonoiu

L3Obis

- **Purpuse** → Alpha version of a flight application software for an On-Board Instrument Control Unit

- **Properties:**

- follows ESA and ECSS standards

- runs on a Leon 2/3 simulated processor

- **Functionality:**

- produce and exchange telemetry packets

- monitor the Instrument and the CDPU status

- acquire scientific data from an Instrument simulator

- perform lossless data compression

L3Obis

- We use ESA TASTE framework

taste

The Assert Set of Tools for Engineering

- We use the following technologies:

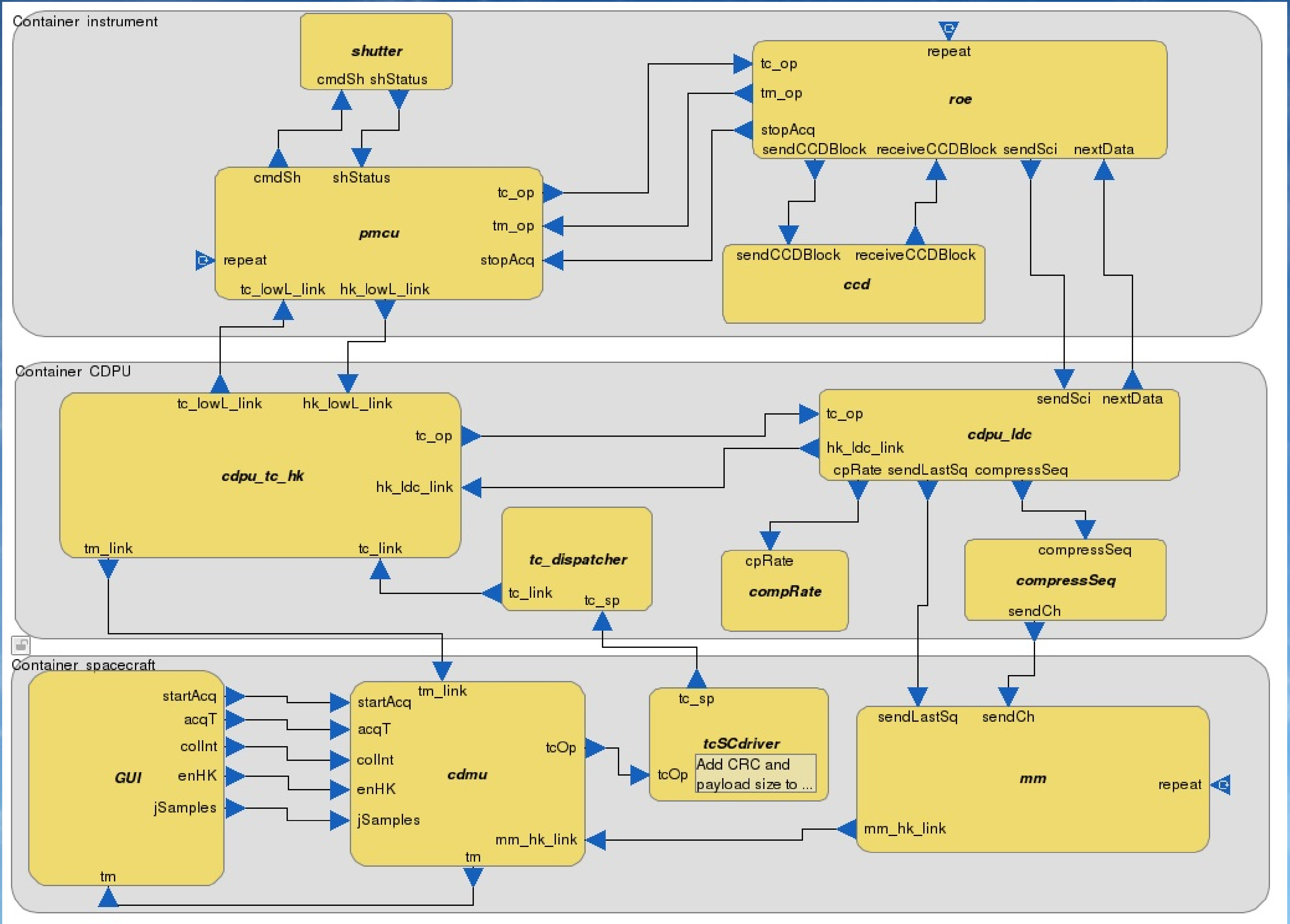
- modeling → ASN1, ACN, AADL, SDL

- code generation & deployment → C

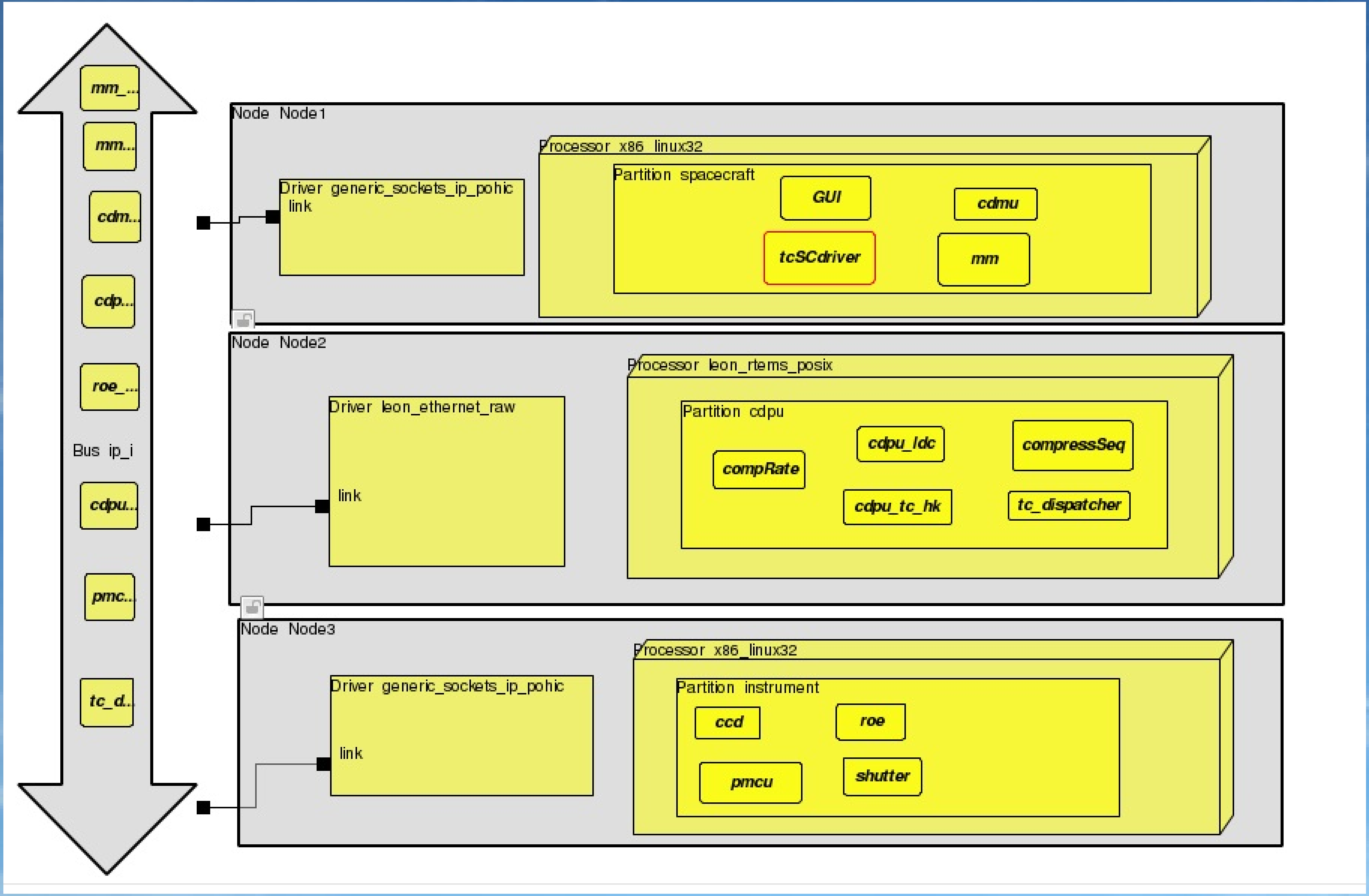
- debugging & testing → MSC

- execution platform → RTEMS, LINUX

L3Obis – Interface View



L3Obis – Deployment View

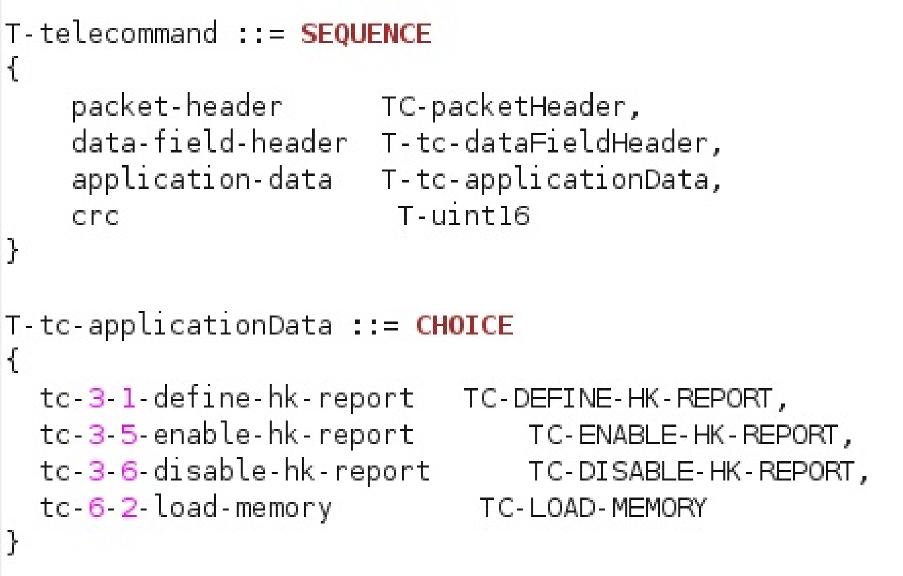


L3Obis

- TC and TM packets → communicate between the components of the system
- ASN1 → DataView.asn → data model
- ACN → DataView.acn → data format

Spare bits → N no. of words

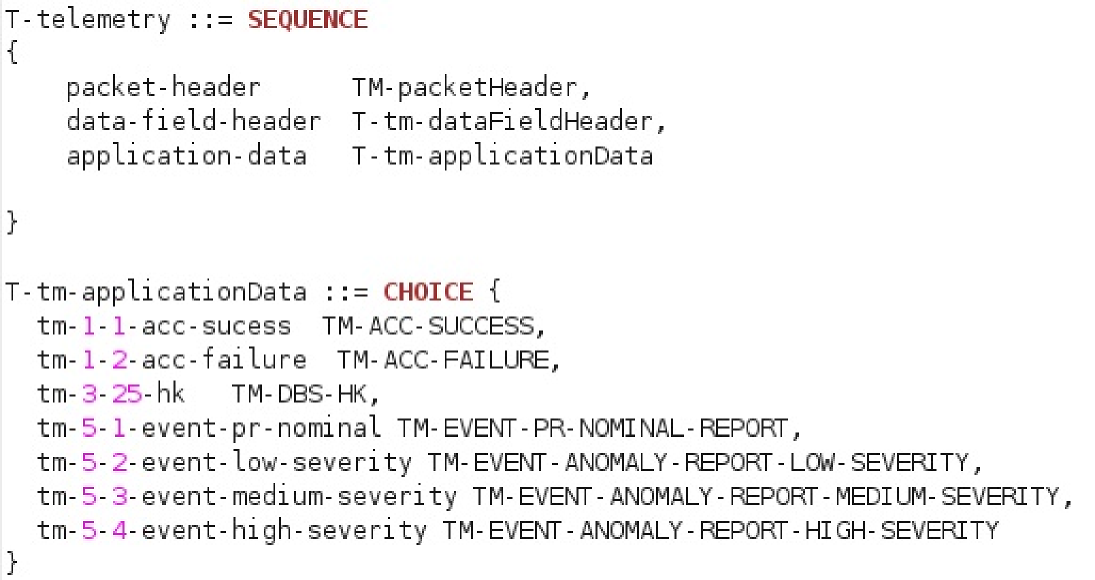
TC packet structure:



```
T-telecommand ::= SEQUENCE
{
    packet-header       TC-packetHeader,
    data-field-header   T-tc-dataFieldHeader,
    application-data    T-tc-applicationData,
    crc                  T-uint16
}

T-tc-applicationData ::= CHOICE
{
  tc-3-1-define-hk-report    TC-DEFINE-HK-REPORT,
  tc-3-5-enable-hk-report        TC-ENABLE-HK-REPORT,
  tc-3-6-disable-hk-report       TC-DISABLE-HK-REPORT,
  tc-6-2-load-memory          TC-LOAD-MEMORY
}
```

TM packet structure:



```
T-telemetry ::= SEQUENCE
{
    packet-header       TM-packetHeader,
    data-field-header   T-tm-dataFieldHeader,
    application-data    T-tm-applicationData

}

T-tm-applicationData ::= CHOICE {
  tm-1-1-acc-sucess   TM-ACC-SUCCESS,
  tm-1-2-acc-failure   TM-ACC-FAILURE,
  tm-3-25-hk    TM-DBS-HK,
  tm-5-1-event-pr-nominal TM-EVENT-PR-NOMINAL-REPORT,
  tm-5-2-event-low-severity TM-EVENT-ANOMALY-REPORT-LOW-SEVERITY,
  tm-5-3-event-medium-severity TM-EVENT-ANOMALY-REPORT-MEDIUM-SEVERITY,
  tm-5-4-event-high-severity TM-EVENT-ANOMALY-REPORT-HIGH-SEVERITY
}
```

- TC and TM packets are set according to ECSS-E-70-41

L3Obis: TC / TM services

TC:

- Load Memory using Absolute Addresses service

  → load data to an area of memory block
- Define new housekeeping reports service

  → set the collection interval
- Enable/Disable Housekeeping Parameter Report Generation service

TM:

- Housekeeping Parameter Report service

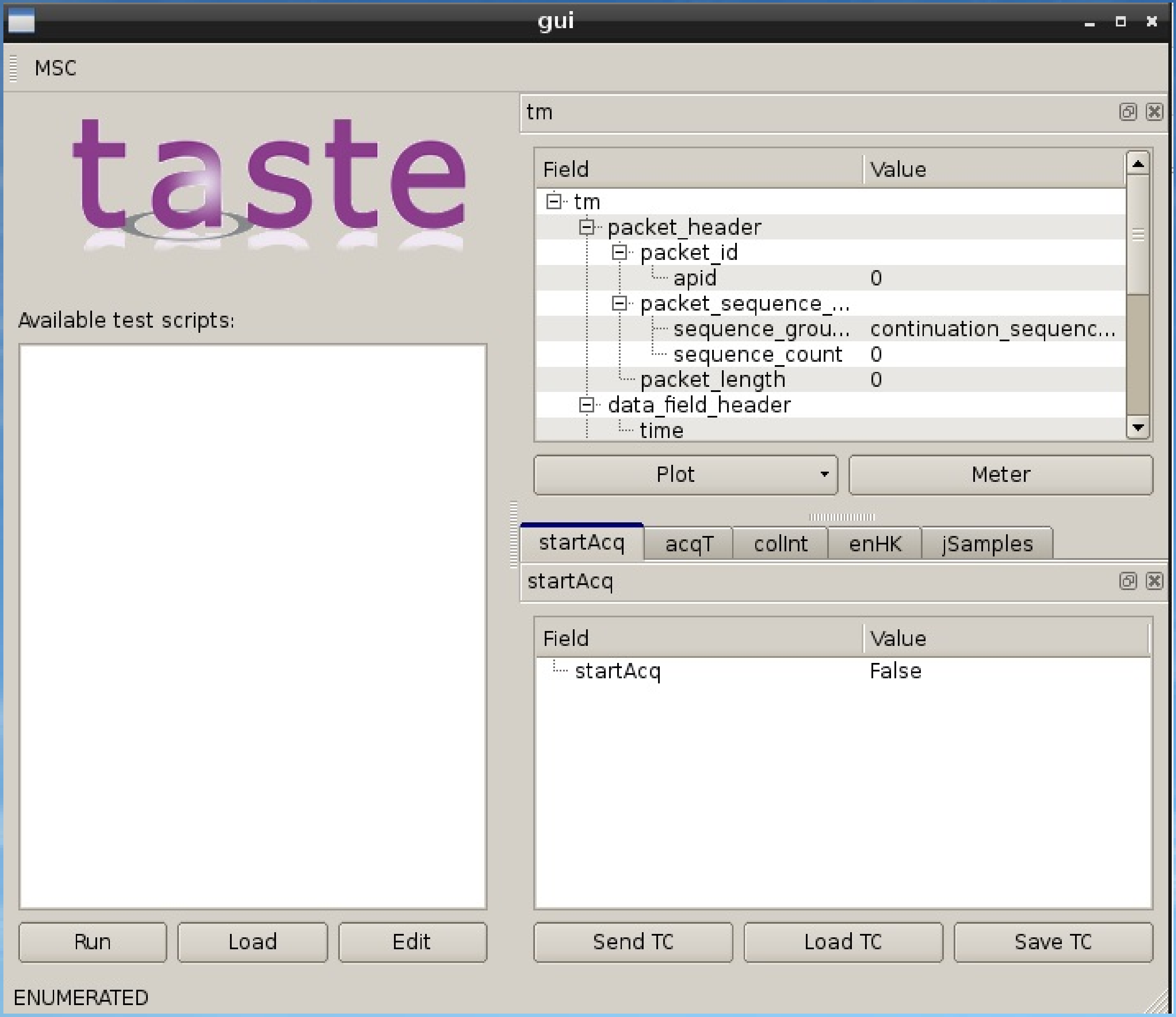
  → report HK parameters
- Telecommand Acceptance Report – Success/Failure

  → verify TC integrity
- Normal Progress/Error Report service

  → link integrity, data storing

L3Obis: Data acquisition process

- Start data acquisition:



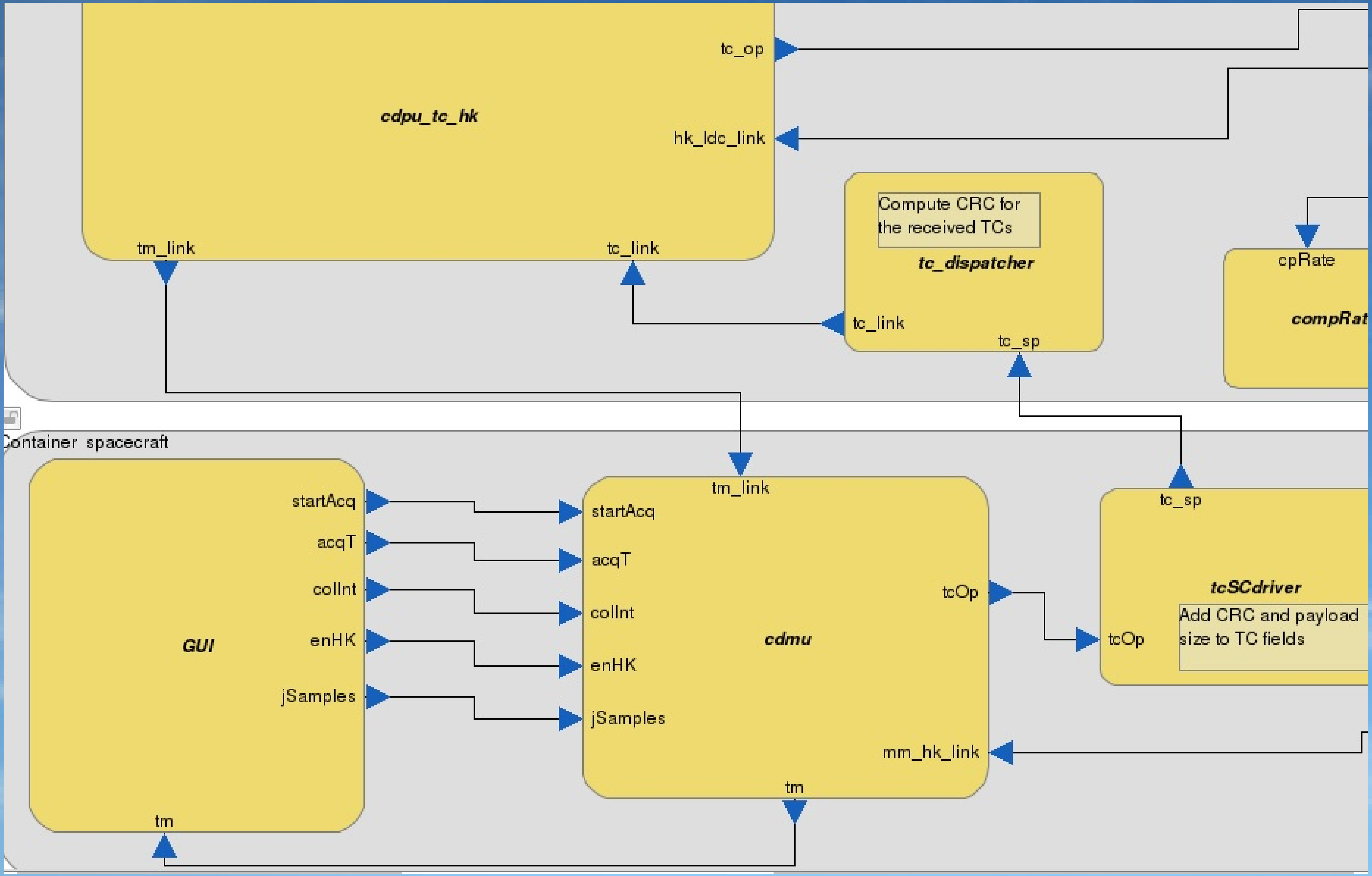
Field	Value
tm	
packet_header	
packet_id	
apid	0
packet_sequence_...	
sequence_grou...	continuation_sequenc...
sequence_count	0
packet_length	0
data_field_header	
time	

Field	Value
startAcq	False

L3Obis: Data acquisition process

- Call *buildTC()* function to construct TC

- Set *process_id*, *sequence_groupingFlag*, *sequence_count*

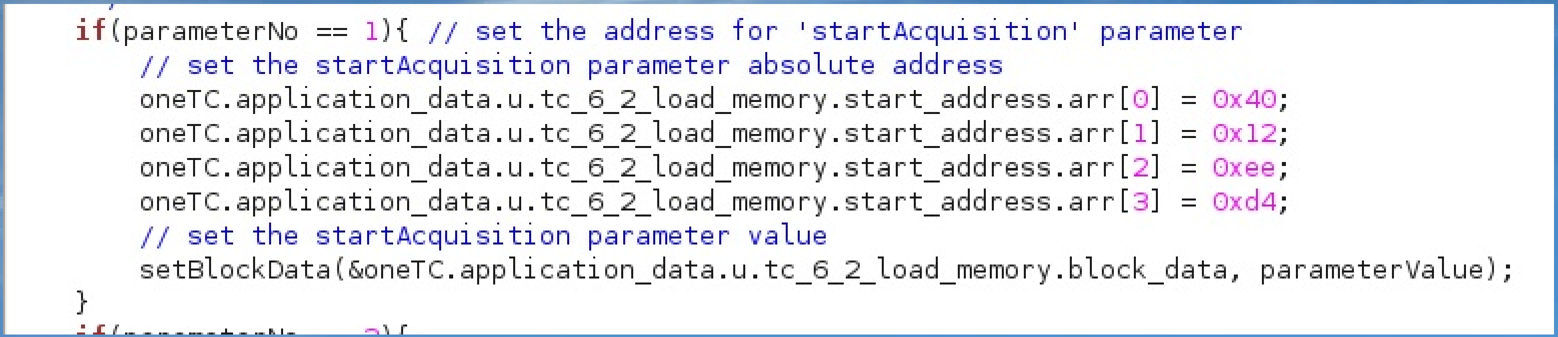


L3Obis: Data acquisition process

- Set the specific TC fields

- Set the Load Memory using A.A. service parameters:

→ *startAcquisition*: { absolute address
value



```
    if(parameterNo == 1){ // set the address for 'startAcquisition' parameter
        // set the startAcquisition parameter absolute address
        oneTC.application_data.u.tc_6_2_load_memory.start_address.arr[0] = 0x40;
        oneTC.application_data.u.tc_6_2_load_memory.start_address.arr[1] = 0x12;
        oneTC.application_data.u.tc_6_2_load_memory.start_address.arr[2] = 0xee;
        oneTC.application_data.u.tc_6_2_load_memory.start_address.arr[3] = 0xd4;
        // set the startAcquisition parameter value
        setBlockData(&oneTC.application_data.u.tc_6_2_load_memory.block_data, parameterValue);
    }
    if(parameterNo == 2){
```

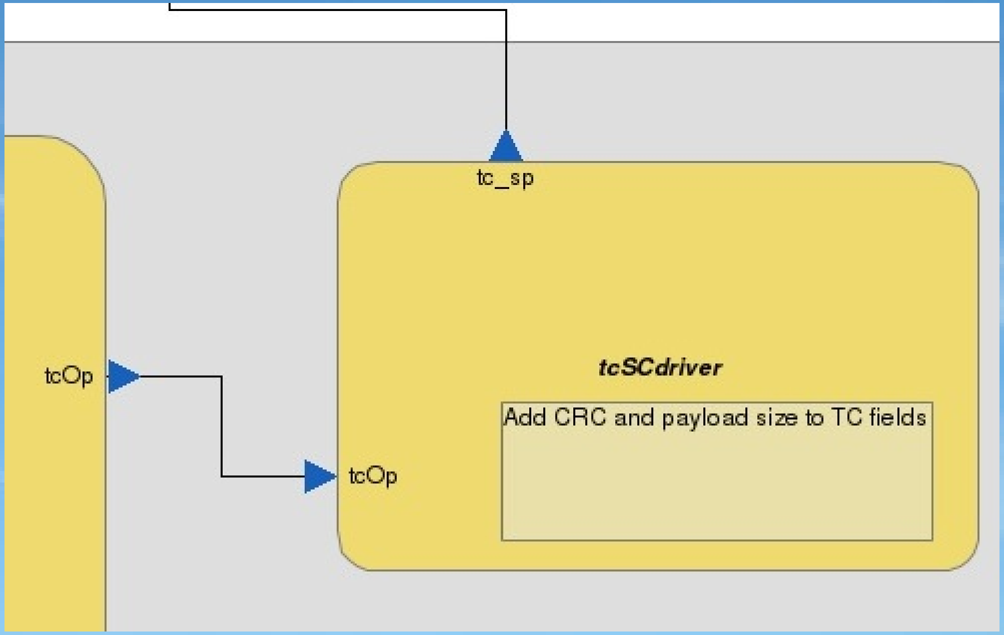
- Send TC to a blackbox device

→ compute Packet Data Field

→ add it to Packet Length

→ compute the checksum

→ add it to the Packet Error Control



Add CRC and payload size to TC fields

L3Obis: Verify TC integrity

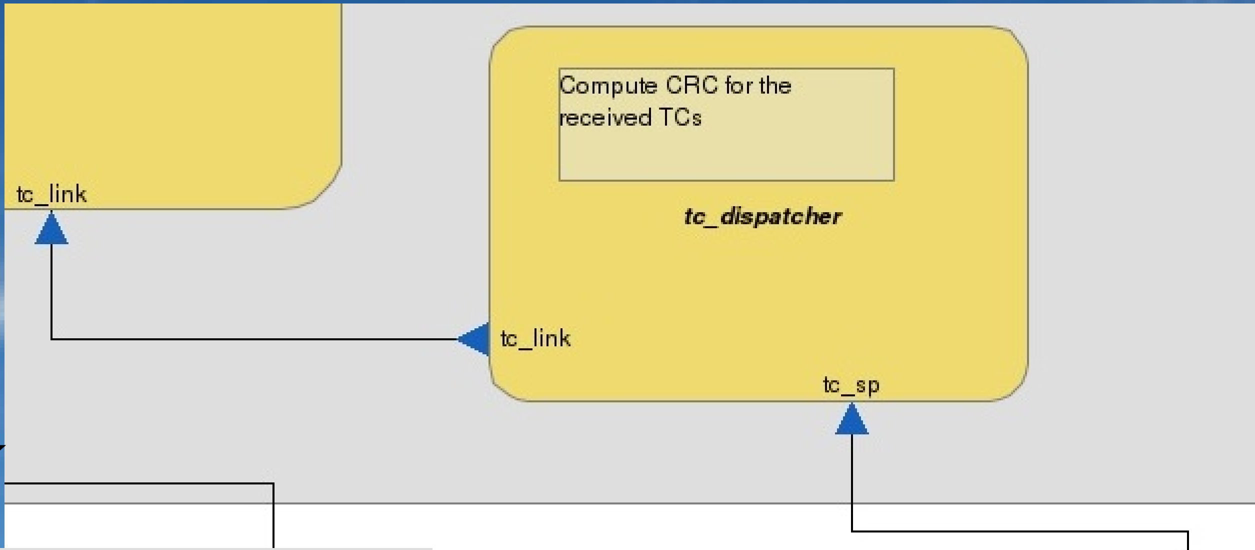
- Sends TC to CDPU

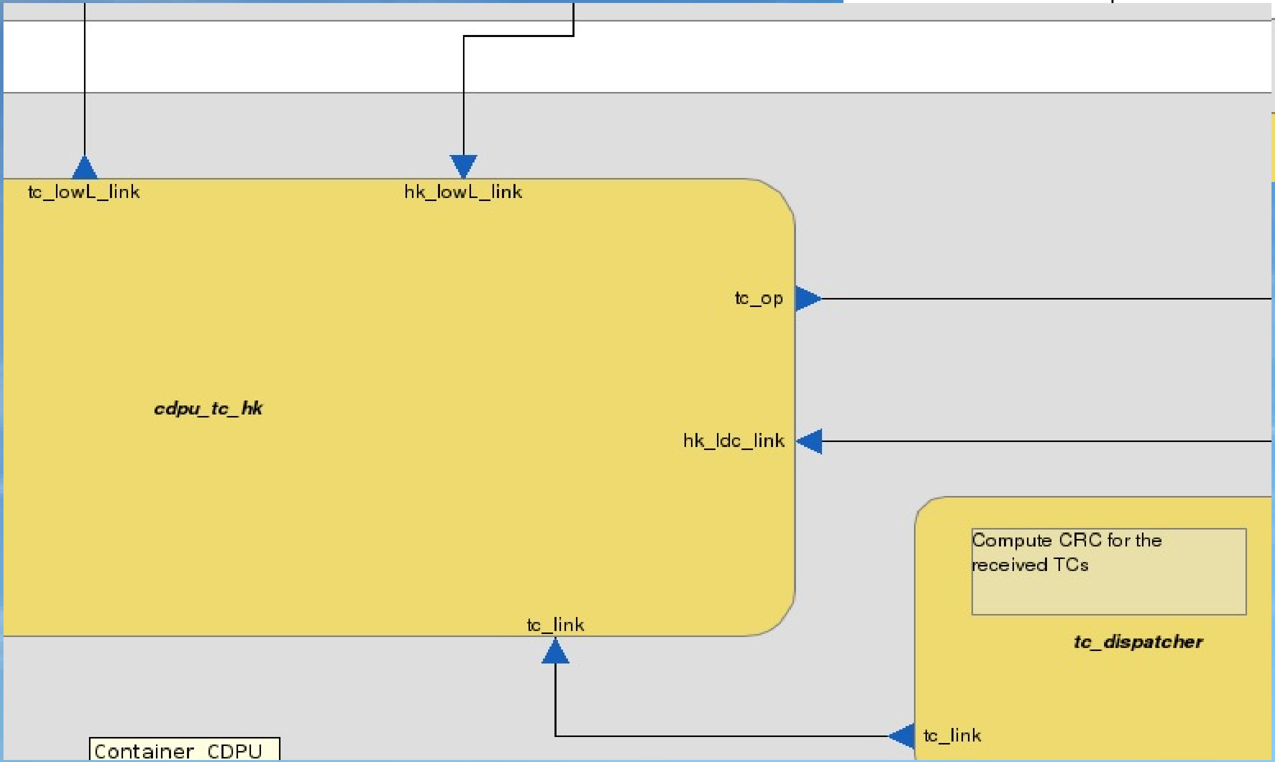
→ *tc_dispatcher*

blackbox_device

→ calculate checksum

→ sends TC to *cdpu_tc_hk*





- read TC Ack field

→ verify TC integrity

L3Obis: Verify TC integrity

- Checksum == 0 → no transmission errors => keep TC

- Use TC Acceptance Report – Success (1, 1) service

→ send TM packet to cdmu

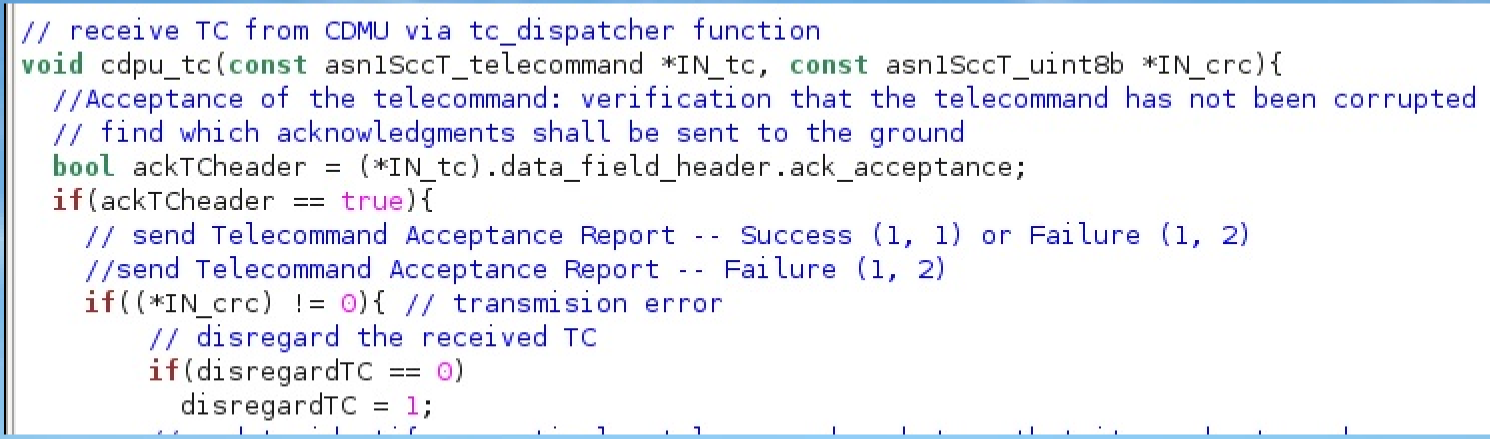
- cdpu_tc_hk_startup() → sets TM general fields

- Checksum != 0

→ use Telecommand Acceptance Report – Failure (1,2)

→ send TM packet to cdmu

→ disregard TC

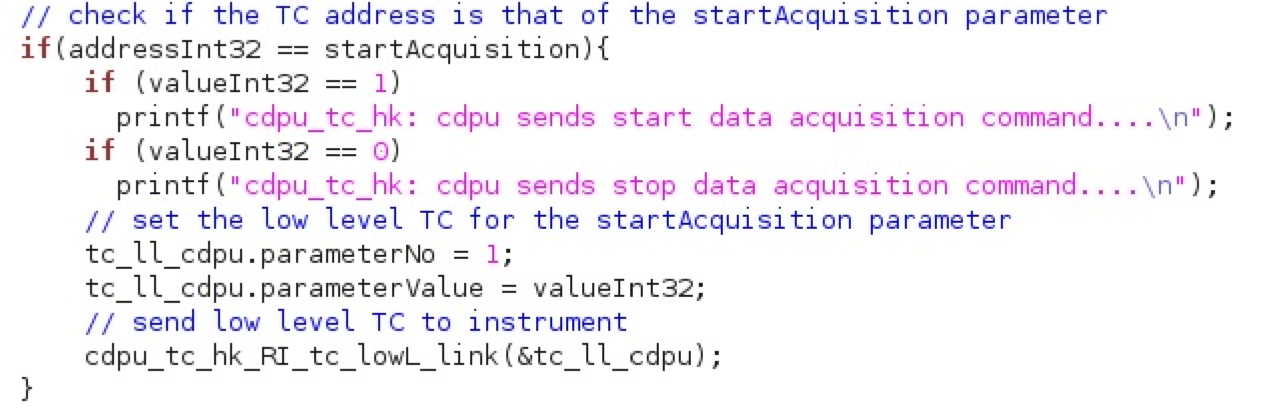


```
// receive TC from CDMU via tc_dispatcher function
void cdpu_tc(const asn1SccT_telecommand *IN_tc, const asn1SccT_uint8b *IN_crc){
  //Acceptance of the telecommand: verification that the telecommand has not been corrupted
  // find which acknowledgments shall be sent to the ground
  bool ackTCheader = (*IN_tc).data_field_header.ack_acceptance;
  if(ackTCheader == true){
    // send Telecommand Acceptance Report -- Success (1, 1) or Failure (1, 2)
    //send Telecommand Acceptance Report -- Failure (1, 2)
    if((*IN_crc) != 0){ // transmision error
        // disregard the received TC
        if(disregardTC == 0)
          disregardTC = 1;
```

L3Obis: Verify TC integrity

- Find the TC service type

→ check the *startAcquisition* parameter address & value



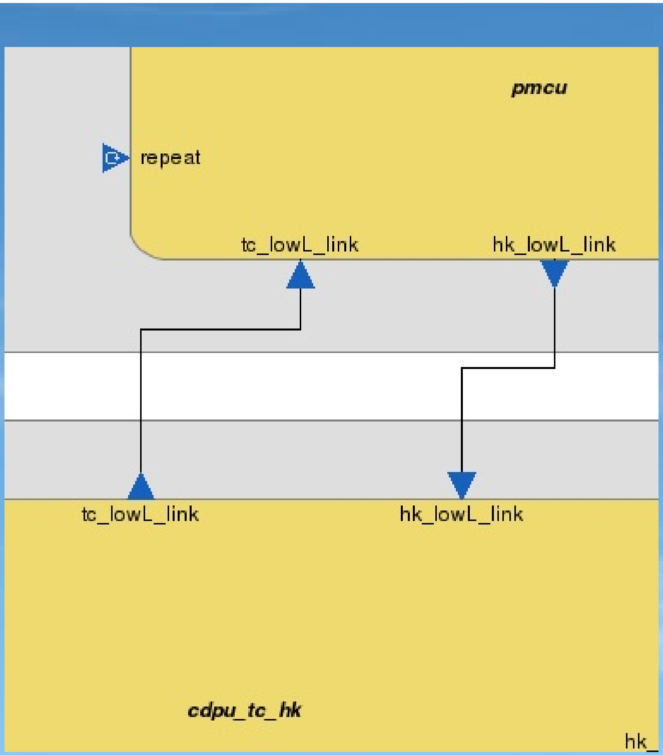
```
// check if the TC address is that of the startAcquisition parameter
if(addressInt32 == startAcquisition){
    if (valueInt32 == 1)
      printf("cdpu_tc_hk: cdpu sends start data acquisition command....\n");
    if (valueInt32 == 0)
      printf("cdpu_tc_hk: cdpu sends stop data acquisition command....\n");
    // set the low level TC for the startAcquisition parameter
    tc_ll_cdpu.parameterNo = 1;
    tc_ll_cdpu.parameterValue = valueInt32;
    // send low level TC to instrument
    cdpu_tc_hk_RI_tc_lowL_link(&tc_ll_cdpu);
}
```

- Build a low level TC

→ Set fields

$\begin{cases} \text{parameterNo} \\ \text{parameterValue} \end{cases}$

- Send low level TC to pmcu



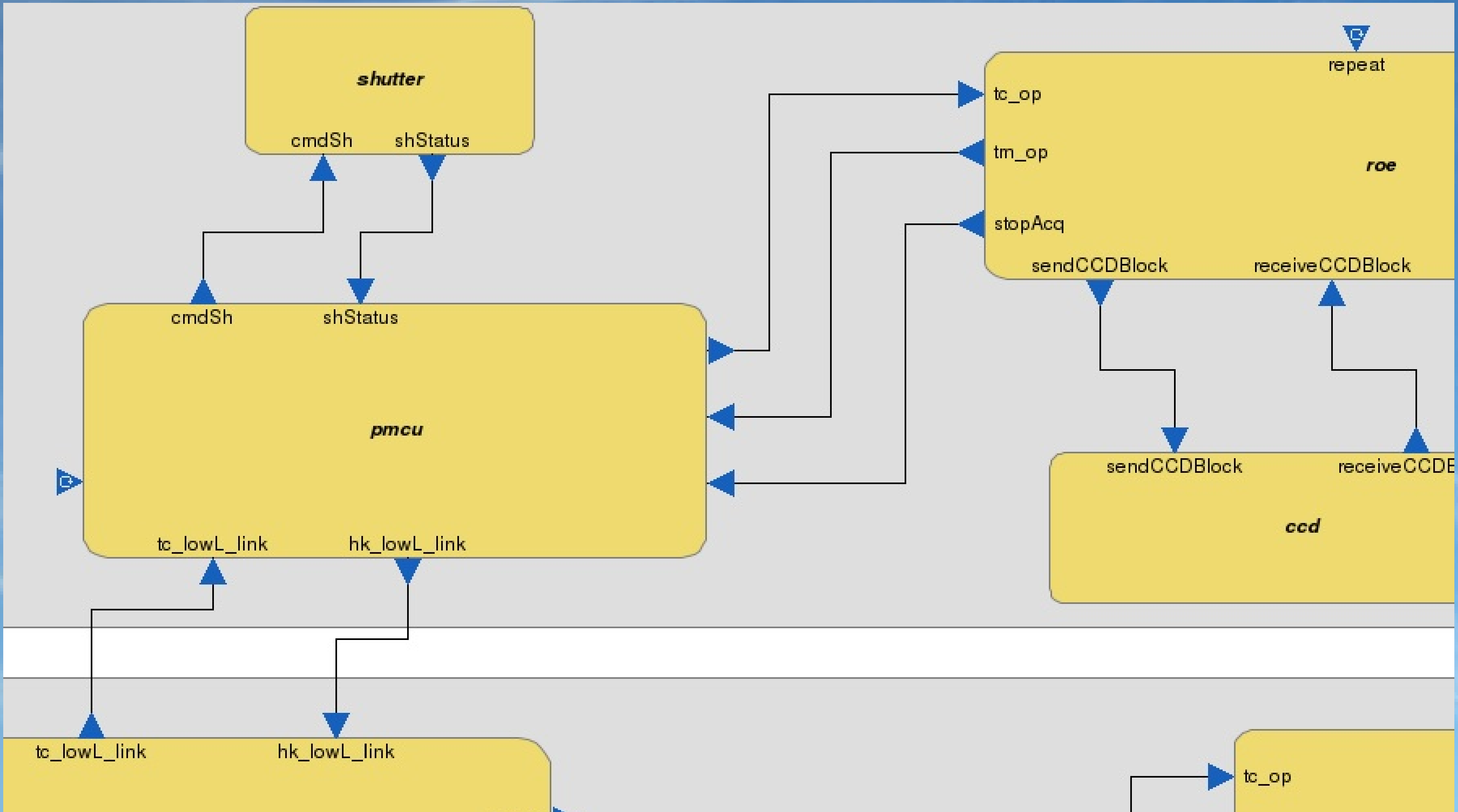
L3Obis: Data acquisition process

- Start data acquisition:

→ *pmcu* commands the shutter to open and the roe to start

- Stop data acquisition:

→ *pmcu* commands the roe to stop and shutter to close



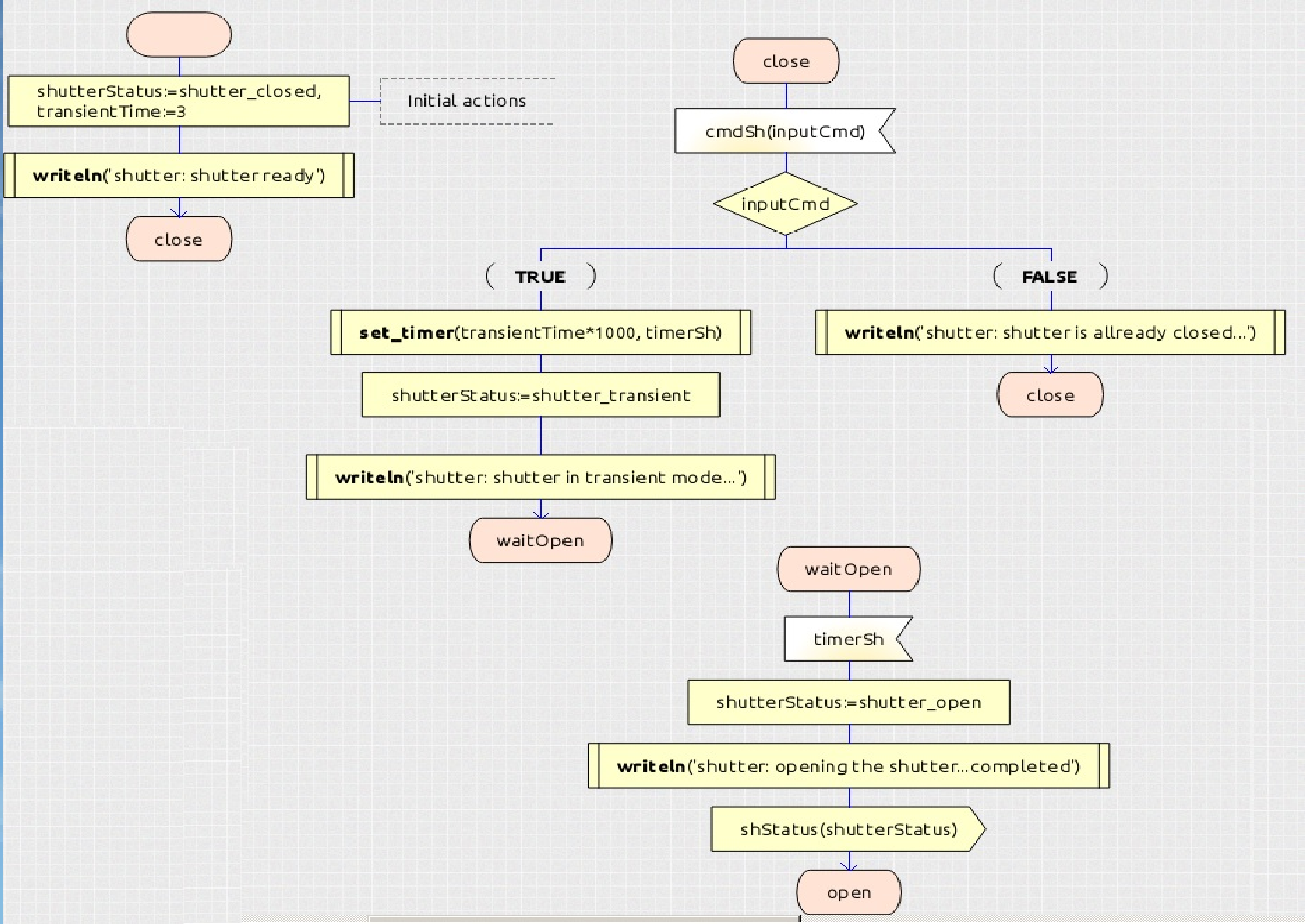
L3Obis: Data acquisition process

- SDL shutter function:

→ open, waitOpen, close, waitClose states

→ *transientTime* → set the transition period

→ report *shutterStatus* to pmcu



shutterStatus:=shutter_closed,
transientTime:=3

Initial actions

writeln('shutter: shutter ready')

close

close

cmdSh(inputCmd)

inputCmd

(TRUE)

set_timer(transientTime*1000, timerSh)

shutterStatus:=shutter_transient

writeln('shutter: shutter in transient mode...')

waitOpen

(FALSE)

writeln('shutter: shutter is allready closed...')

close

waitOpen

timerSh

shutterStatus:=shutter_open

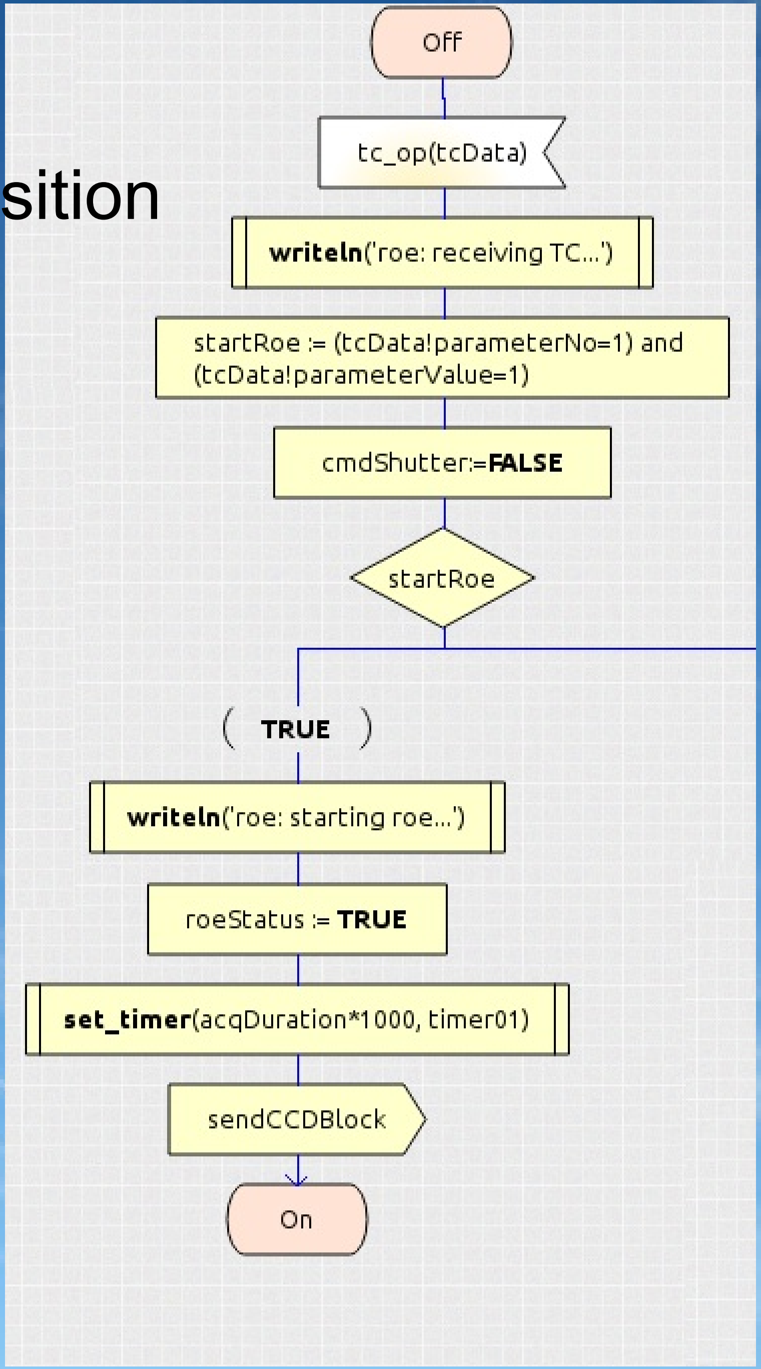
writeln('shutter: opening the shutter...completed')

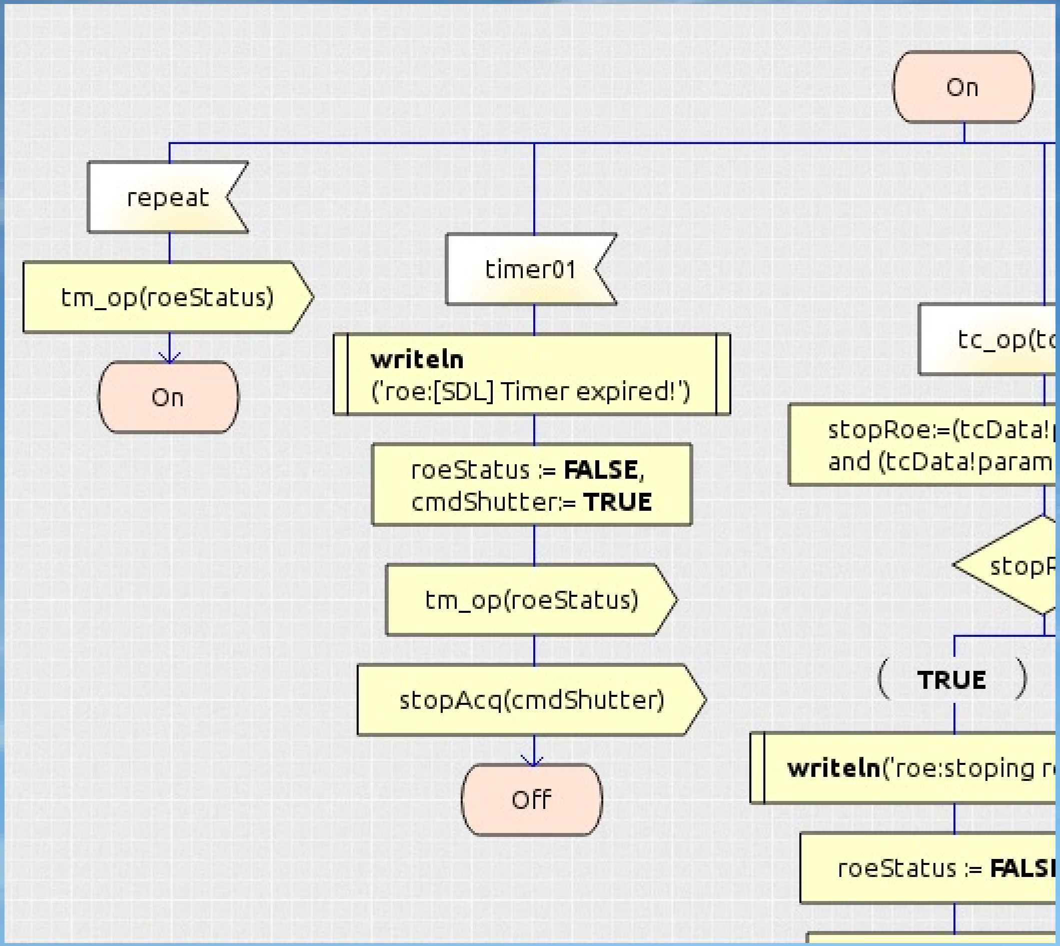
shStatus(shutterStatus)

open

L3Obis: Data acquisition process - ROE

- Upon TC goes from Off to On state
- *acqDuration* → set timer for data acquisition
- call *stopAcq* PI


Off
tc_op(tcData)
writeln('roe: receiving TC...')
startRoe := (tcData!parameterNo=1) and (tcData!parameterValue=1)
cmdShutter:=FALSE
startRoe
(TRUE)
writeln('roe: starting roe...')
roeStatus := TRUE
set_timer(acqDuration*1000, timer01)
sendCCDBlock
On



L3Obis: Data acquisition process

- TASTE GUI: *acqT* parameter

→ set the time interval for the data acquisition

- Sent TC to *cdpu*:

→ *buildTC*()

→ type (6,2) TC service

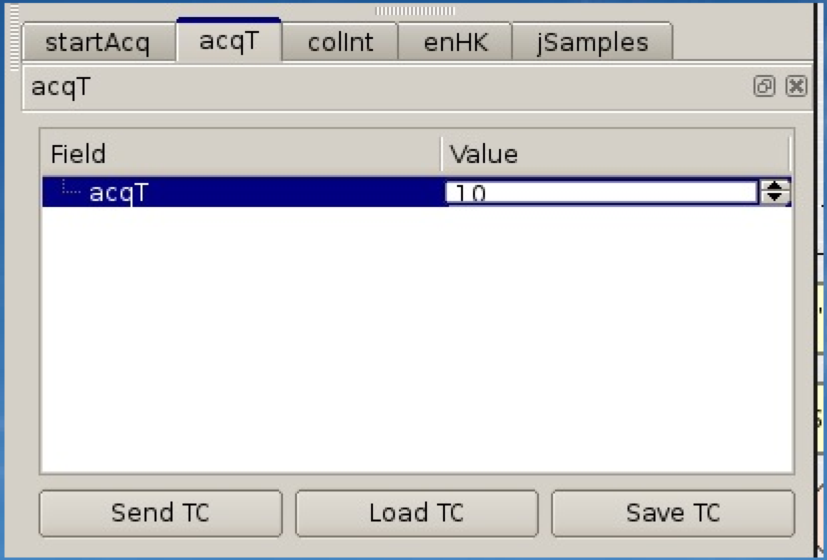
- *cdpu*:

→ check TC integrity

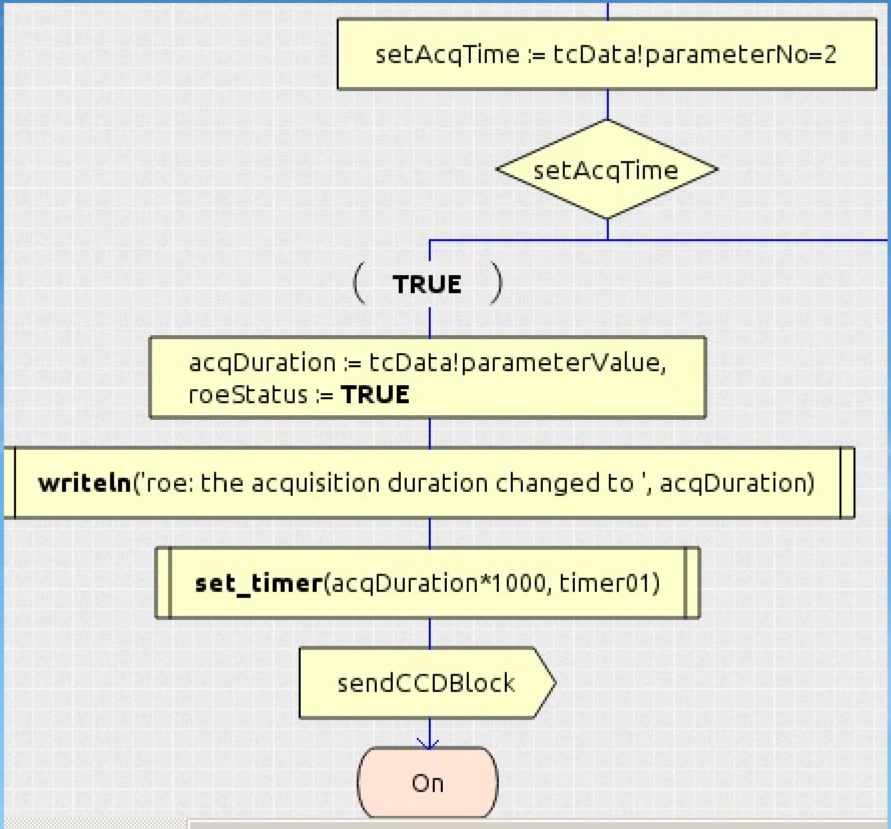
→ route to *instrument* app.

- roe:

→ set *timer01*



Field	Value
acqT	10



```
setAcqTime := tcData!parameterNo=2
setAcqTime
( TRUE )
acqDuration := tcData!parameterValue,
roeStatus := TRUE
writeln('roe: the acquisition duration changed to ', acqDuration)
set_timer(acqDuration*1000, timer01)
sendCCDBlock
On
```

L3Obis: Data acquisition process

- Call *ccd* function

→ *sendCCDBlock*

- *ccd*:

→ reads from input file

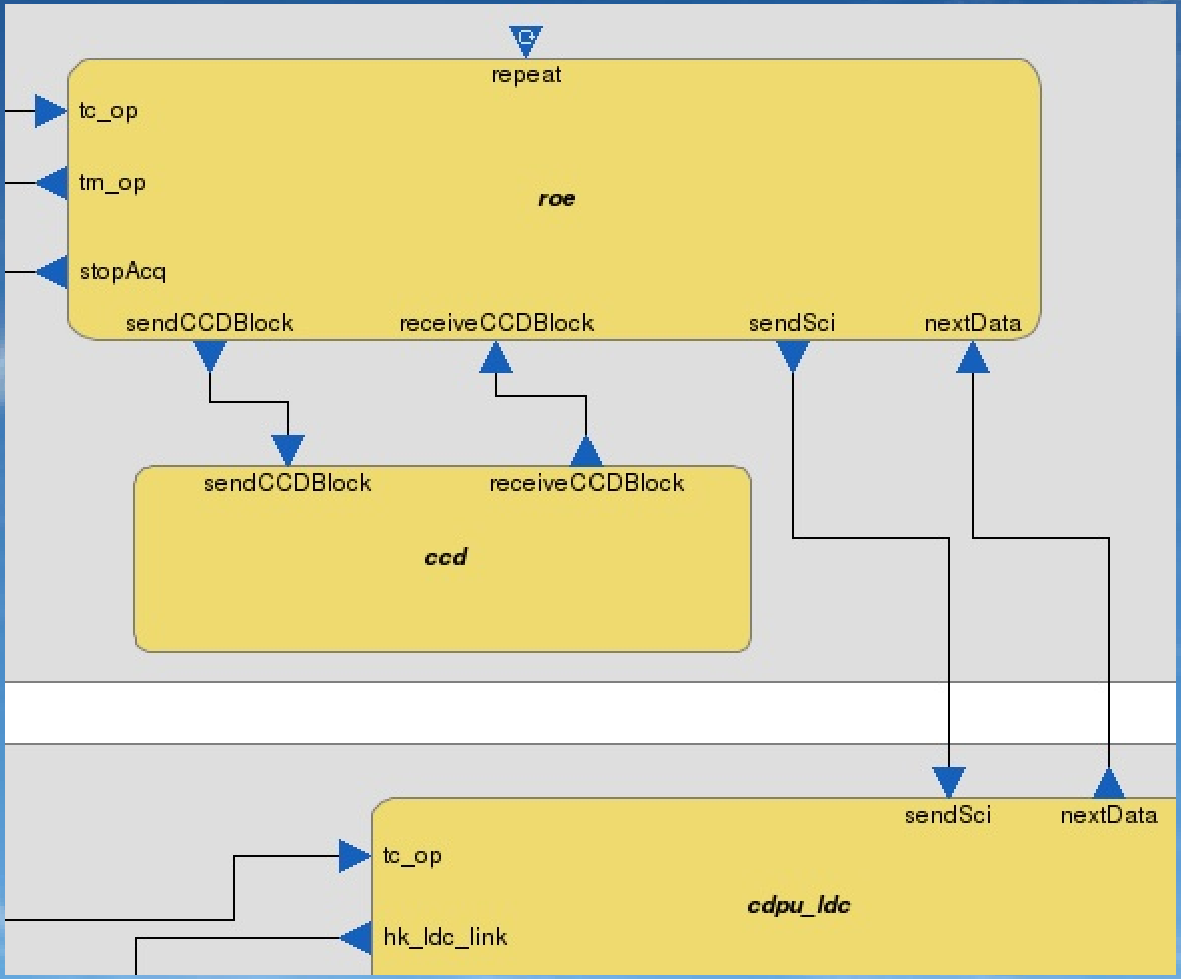
#define sequenceNoOfChars 1024

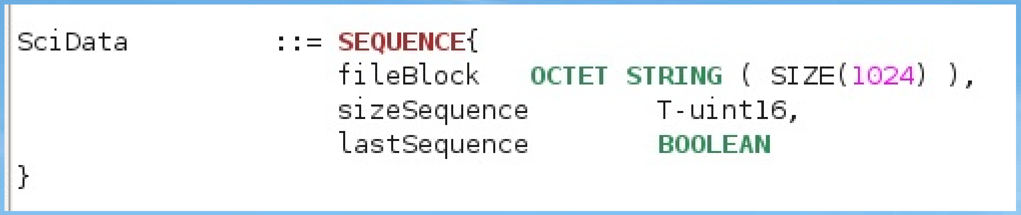
- Call *roe* function

→ *receiveCCDBlock*

- SciData structure:

→ DataView.asn





```
SciData          ::= SEQUENCE{
                      fileBlock    OCTET STRING ( SIZE(1024) ),
                      sizeSequence        T-uint16,
                      lastSequence        BOOLEAN
}
```

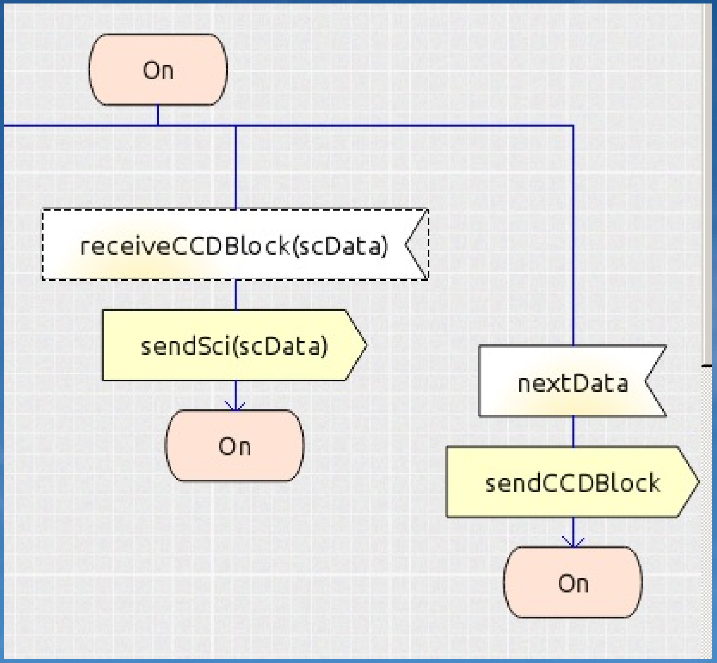
L3Obis: Data acquisition process

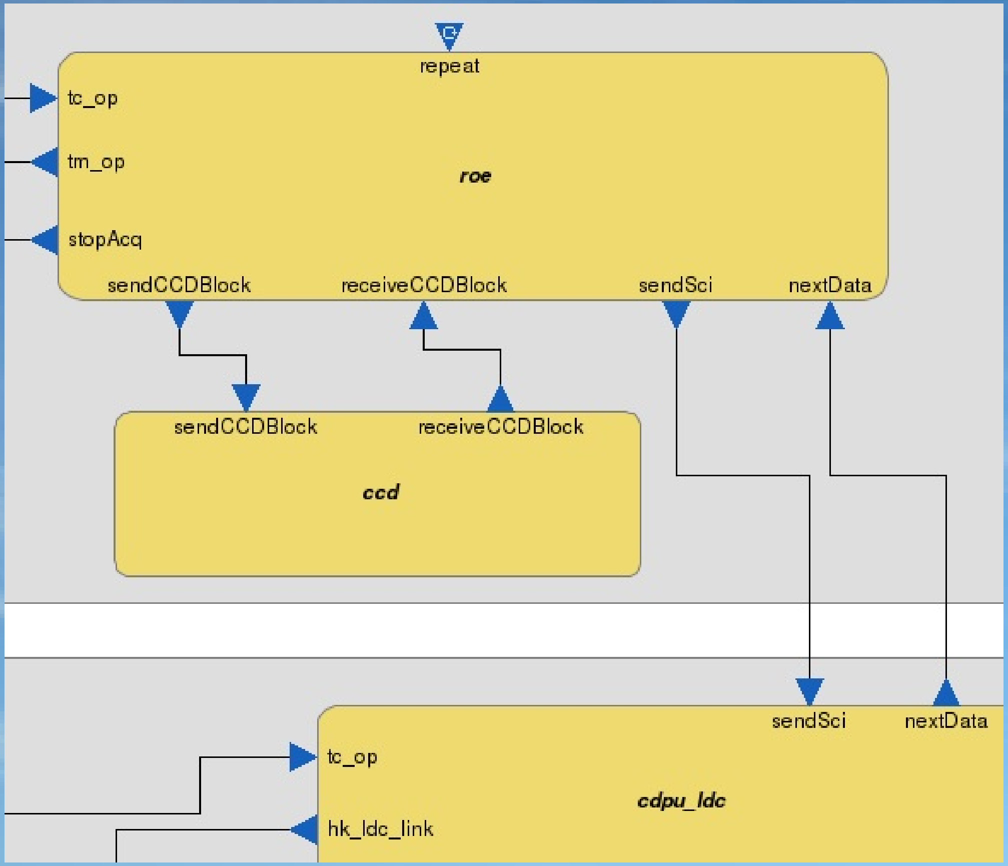
- CCD *sciData*

→ *roe* On state

- send *sciData* to compression:

→ *cdpu_ldc* function





- cdpu_ldc:

→ call *nextData* interface

- roe: send more data

→ call *sendCCDBlock*

- stop conditions:

→ put *startAcq* to 0

→ *timer01* expires

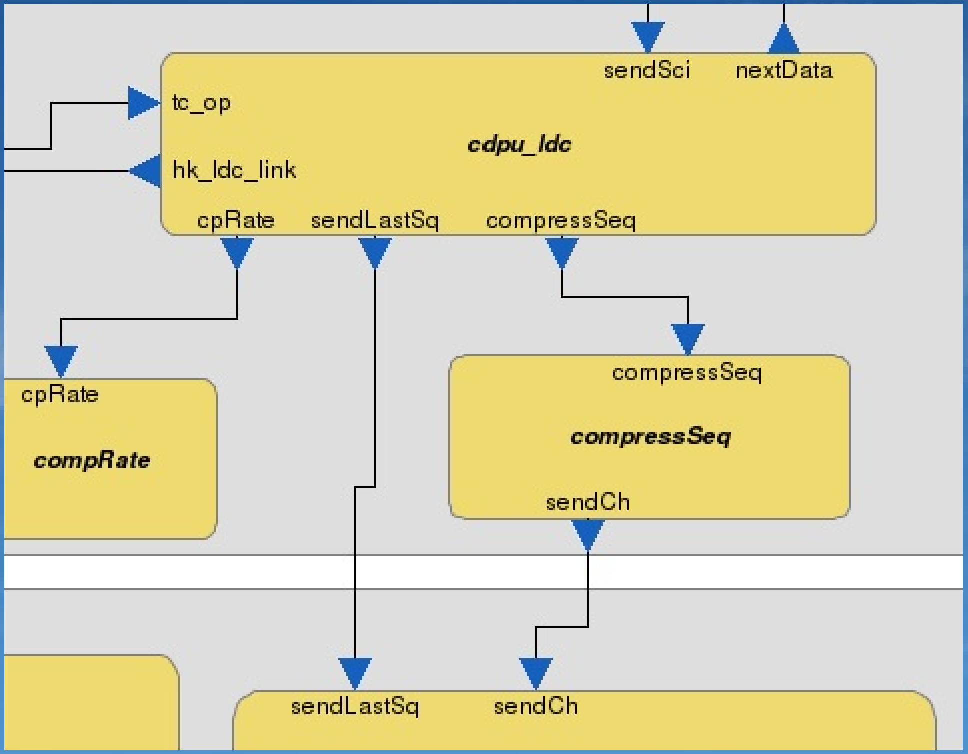
L3Obis: Data compression process

- cdpu_ldc → *sciData*

→ *sendSci* PI

- *compressSeq* → sciData → compressSeq PI:

→ protected interface: new data only after previous block was processed



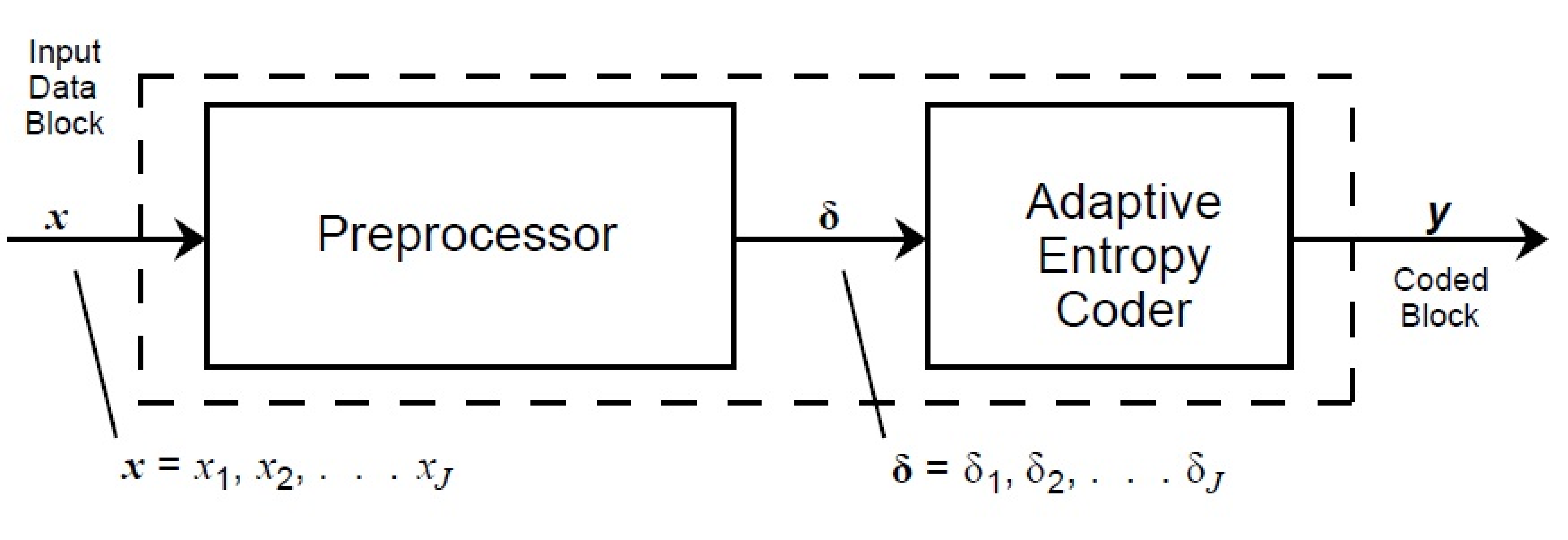
- ccd last incomplete sequence:

→ forward to *sendLastSq* PI

→ store to mm

L3Obis: Rice algorithm

- lossless coding method → preserves the original data accuracy


Input Data Block
$\mathbf{x}$
Preprocessor
δ
Adaptive Entropy Coder
$\mathbf{y}$
Coded Block
$\mathbf{x} = x_1, x_2, \ldots x_J$
$\boldsymbol{\delta} = \delta_1, \delta_2, \ldots \delta_J$

[from CCSDS 121.0-B-2]

- Input block no of samples → J = 8, 16, 32, 64
- AEC → smallest average no of bits for input sample

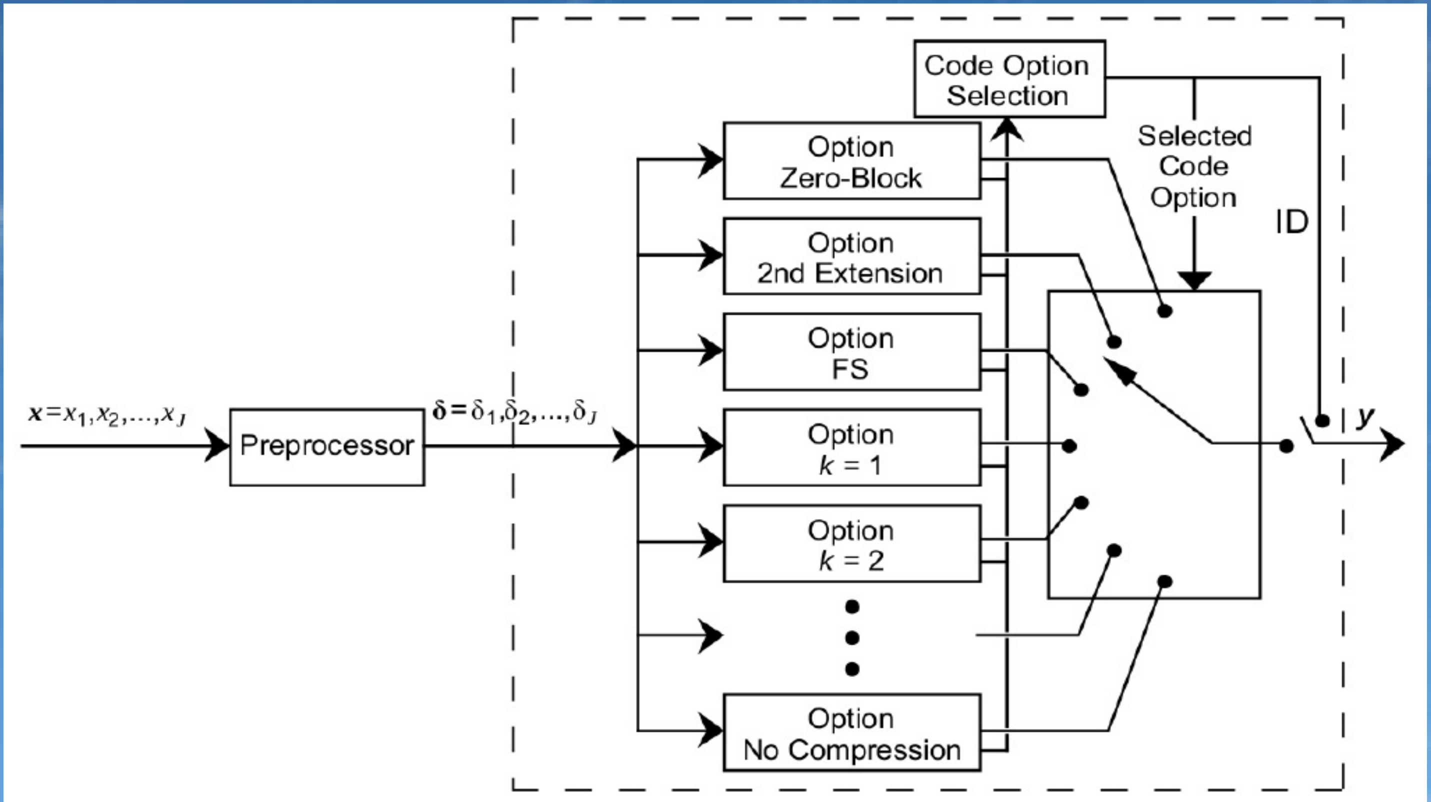
L3Obis: Rice algorithm

- Preprocessor output data

→ AEC module → variable-length codewords

- AEC:

→ sequence of coding options



[from CCSDS 121.0-B-2]

- Select the algorithm option that use the shortest no of bits
- Add the algorithm option ID

→ helps decoder to identify the selected option

L3Obis: Data compression process

- Encapsulate encoded data:

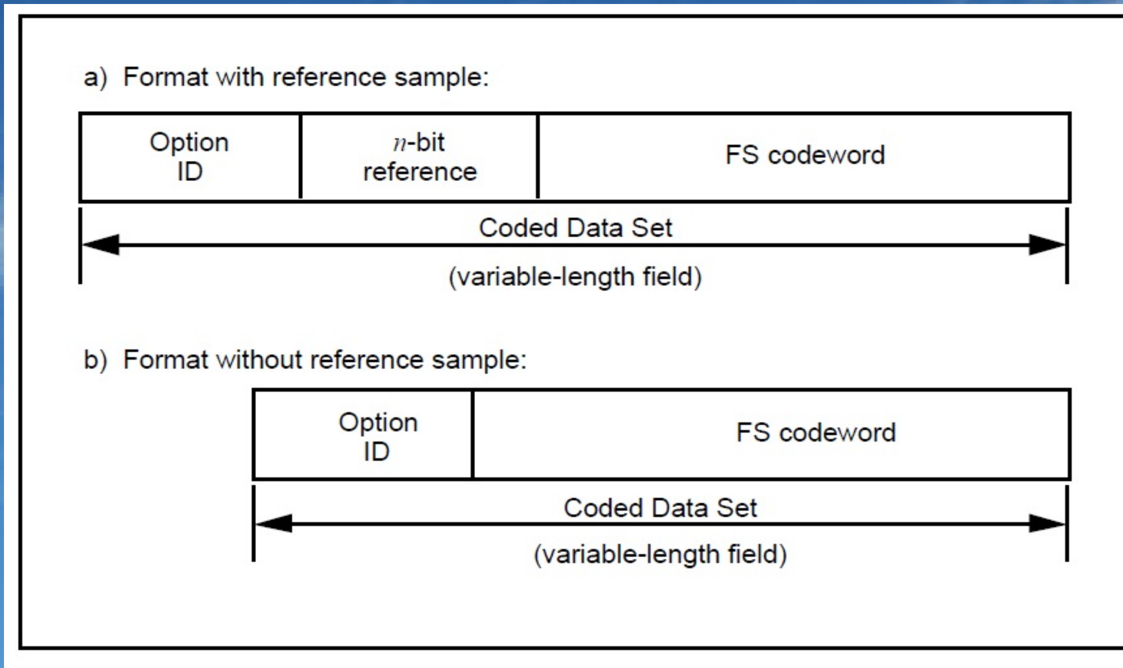
→ CDS format

- AEC output

→ sequence of CDS blocks

- first CDS:

→ add reference sample



a) Format with reference sample:

b) Format without reference sample:

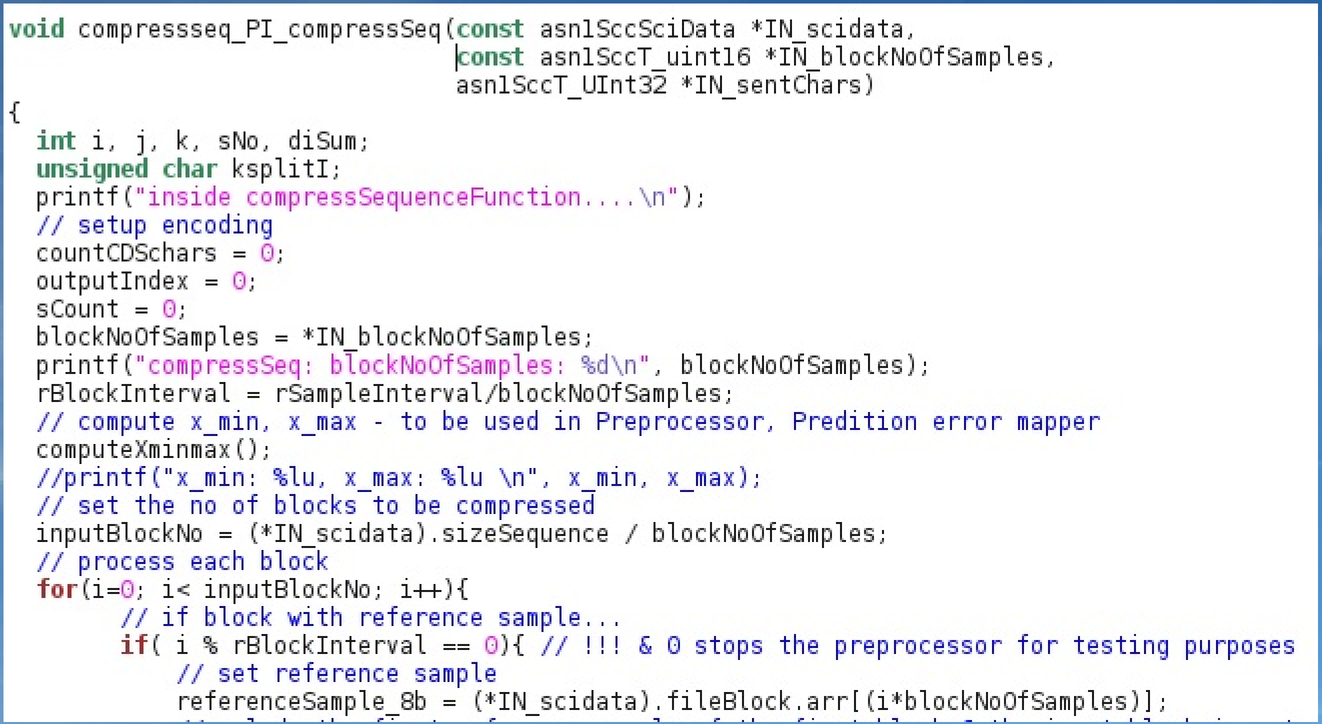
- reference sample: an uncompressed data sample
- *r* = reference sample interval:

→ add periodically a reference sample in the CDS structure

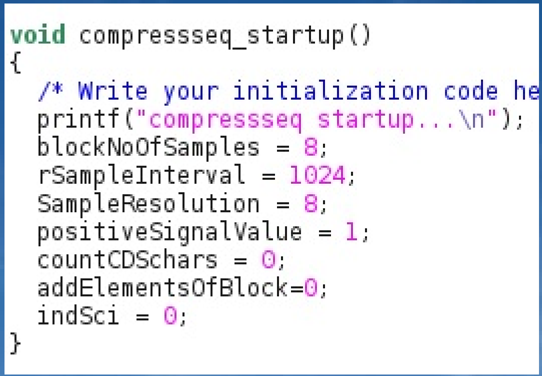
L3Obis: Data compression process

- *compressSeq* function

→ initialize the compression parameters



```c
void compressseq_PI_compressSeq(const asn1SccSciData *IN_scidata,
                                const asn1SccT_uint16 *IN_blockNoOfSamples,
                                asn1SccT_UInt32 *IN_sentChars)
{
  int i, j, k, sNo, diSum;
  unsigned char ksplitI;
  printf("inside compressSequenceFunction....\n");
  // setup encoding
  countCDSchars = 0;
  outputIndex = 0;
  sCount = 0;
  blockNoOfSamples = *IN_blockNoOfSamples;
  printf("compressSeq: blockNoOfSamples: %d\n", blockNoOfSamples);
  rBlockInterval = rSampleInterval/blockNoOfSamples;
  // compute x_min, x_max - to be used in Preprocessor, Predition error mapper
  computeXminmax();
  //printf("x_min: %lu, x_max: %lu \n", x_min, x_max);
  // set the no of blocks to be compressed
  inputBlockNo = (*IN_scidata).sizeSequence / blockNoOfSamples;
  // process each block
  for(i=0; i< inputBlockNo; i++){
        // if block with reference sample...
        if( i % rBlockInterval == 0){ // !!! & 0 stops the preprocessor for testing purposes
            // set reference sample
            referenceSample_8b = (*IN_scidata).fileBlock.arr[(i*blockNoOfSamples)];
```



```c
void compressseq_startup()
{
  /* Write your initialization code he
  printf("compressseq startup...\n");
  blockNoOfSamples = 8;
  rSampleInterval = 1024;
  SampleResolution = 8;
  positiveSignalValue = 1;
  countCDSchars = 0;
  addElementsOfBlock=0;
  indSci = 0;
}
```

- Set *blockNoOfSamples* → (*IN_blockNoOfSamples)
- compute *rBlockInterval*
- Process each input block

- There are two cases:

→ the first sample of the block is a reference sample

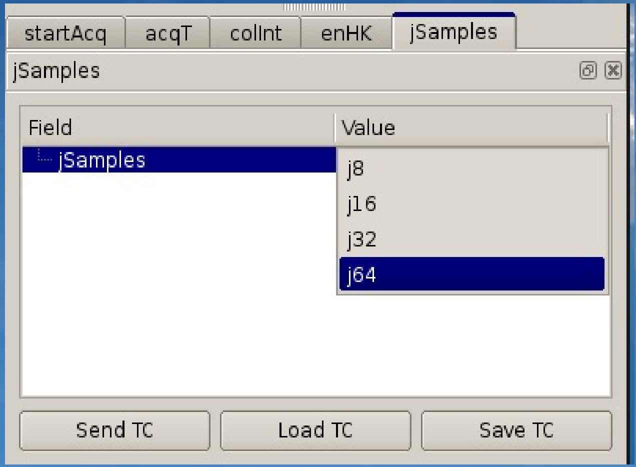
→ the block has no reference sample

- Preprocess the data → *preprocessor_8b* function
- Calculate the no. of bits for each compression option → *selectBestCodingOption*()
- Build and send CDS packet

L3Obis: Data compression process

- Change the no. of samples of the input data block

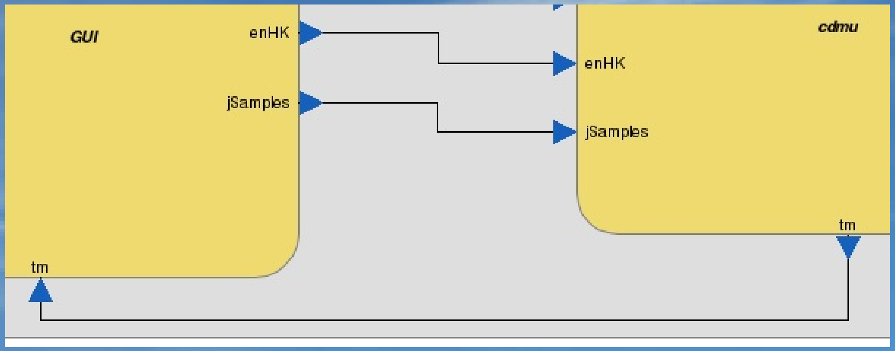
→ select from GUI the no of samples



jSamples

Field	Value
jSamples	j8 j16 j32 j64

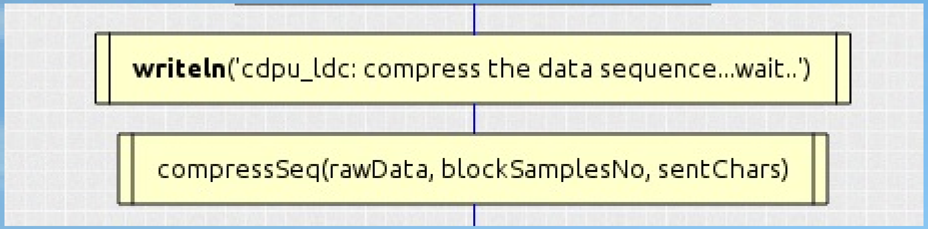
→ GUI → *cdmu* → *buildTC()* → TC (6,2)

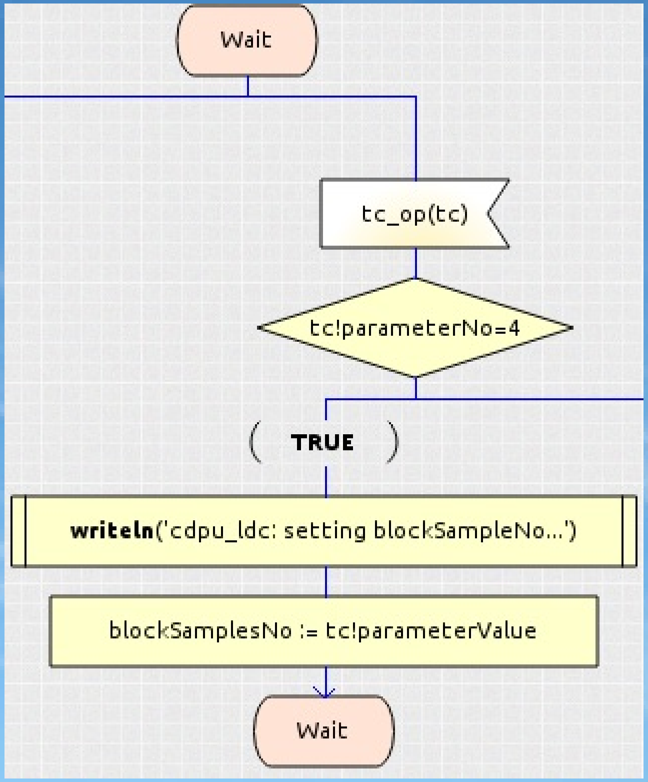


→ cdpu_tc_hk → cdpu_ldc → tc_op PI

→TC sets *blockSamplesNo* variable

→ call *compressSeq*()

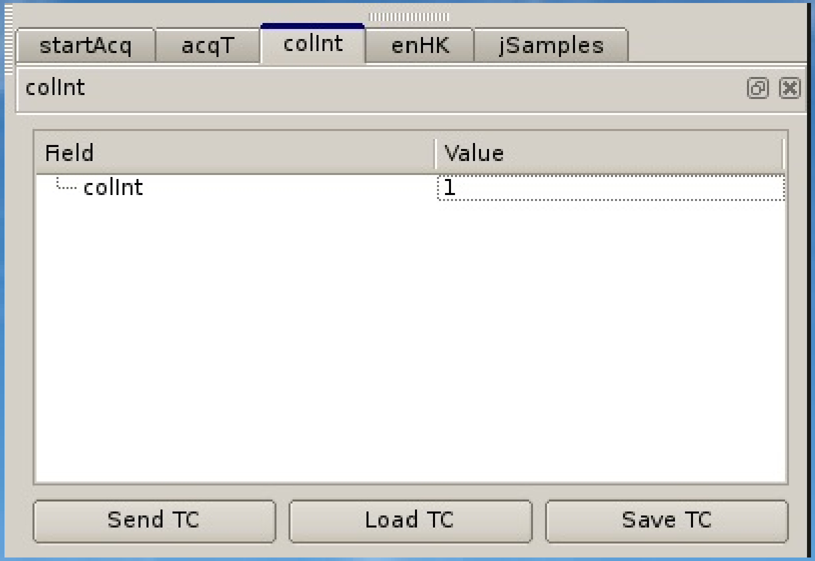

writeln('cdpu_ldc: compress the data sequence...wait..')
compressSeq(rawData, blockSamplesNo, sentChars)


Wait
tc_op(tc)
tc!parameterNo=4
TRUE
writeln('cdpu_ldc: setting blockSampleNo...')
blockSamplesNo := tc!parameterValue
Wait

L3Obis: HK report

- Sends periodic HK:
  - roeStatus
  - shutterStatus
  - compressionRate
  - compressionStatus

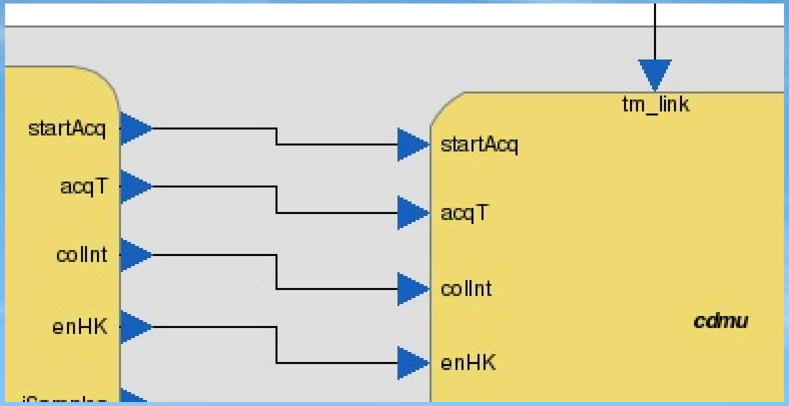
- set T for HK reports
  → use *colInt*



Field	Value
colInt	1

- colInt PI → buildTC

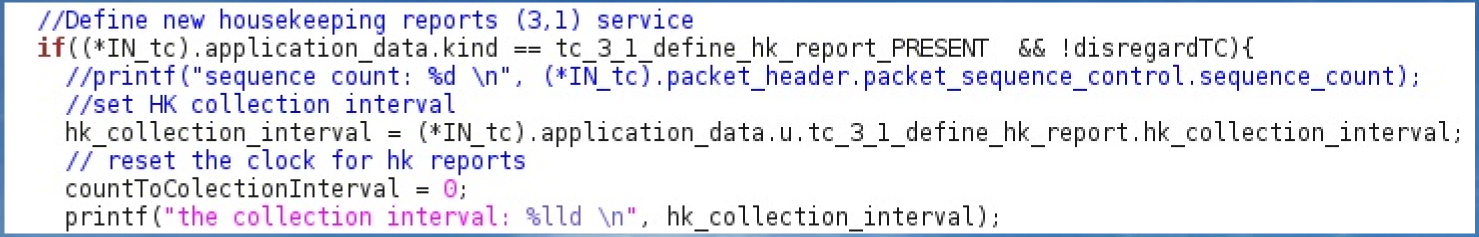
→ TC of type (3,1)



L3Obis: HK report

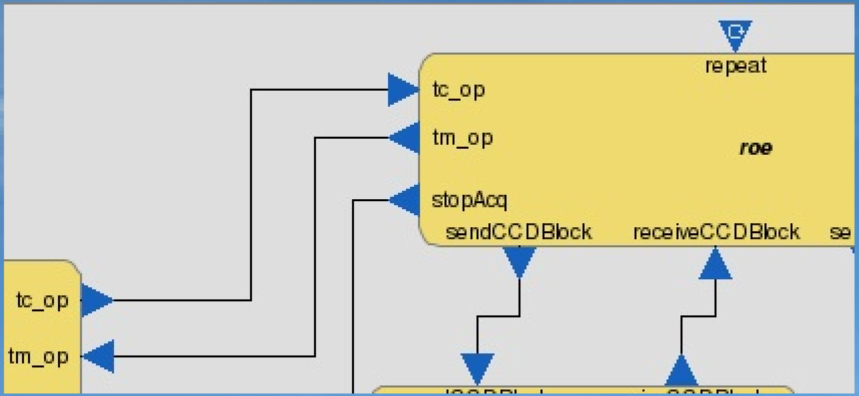
- cdpu_tc_hk function:

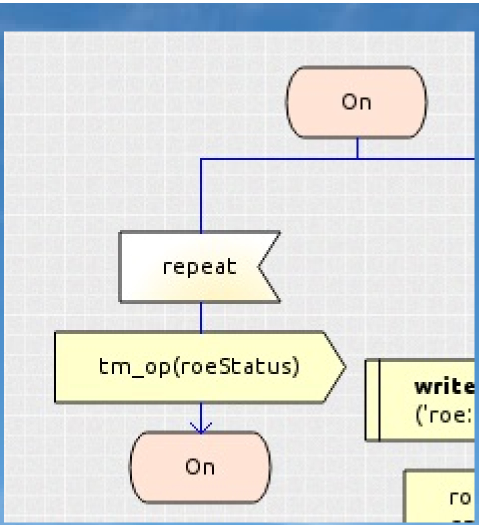
- set *hk_collection_interval* from TC → send HK periodically



```
//Define new housekeeping reports (3,1) service
if((*IN_tc).application_data.kind == tc_3_1_define_hk_report_PRESENT  && !disregardTC){
  //printf("sequence count: %d \n", (*IN_tc).packet_header.packet_sequence_control.sequence_count);
  //set HK collection interval
  hk_collection_interval = (*IN_tc).application_data.u.tc_3_1_define_hk_report.hk_collection_interval;
  // reset the clock for hk reports
  countToColectionInterval = 0;
  printf("the collection interval: %lld \n", hk_collection_interval);
```

- roe → roeStatus → pmcu





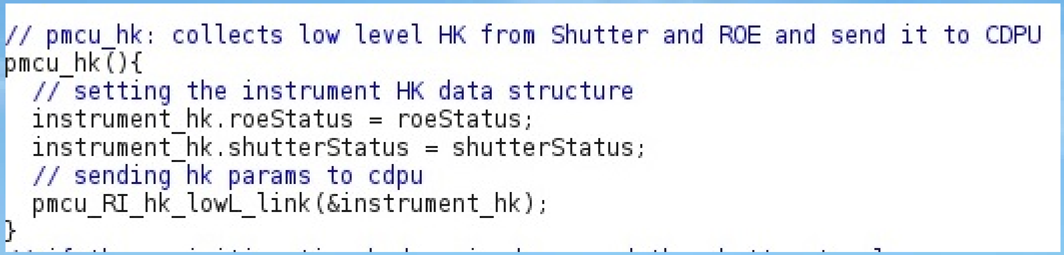
- *roe* State: On → Off => *roeStatus* → FALSE

  Off → On => *roeStatus* → TRUE

- *pmcu* → low level HK:

{ *roeStatus*, *shutterStatus* }

→ *cdpu_tc_hk*



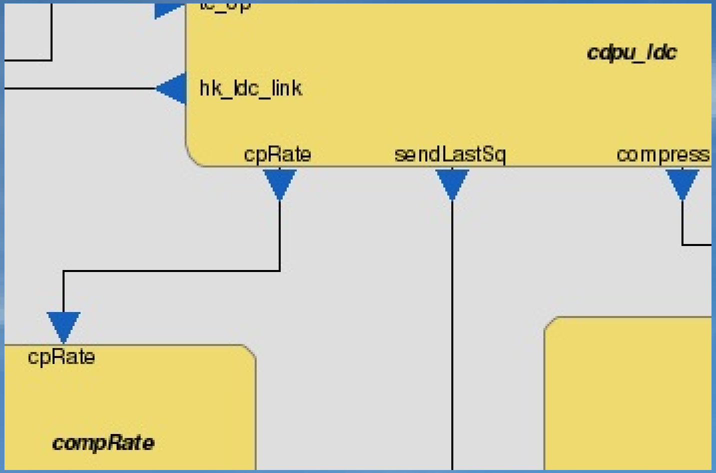
```
// pmcu_hk: collects low level HK from Shutter and ROE and send it to CDPU
pmcu_hk(){
  // setting the instrument HK data structure
  instrument_hk.roeStatus = roeStatus;
  instrument_hk.shutterStatus = shutterStatus;
  // sending hk params to cdpu
  pmcu_RI_hk_lowL_link(&instrument_hk);
}
```

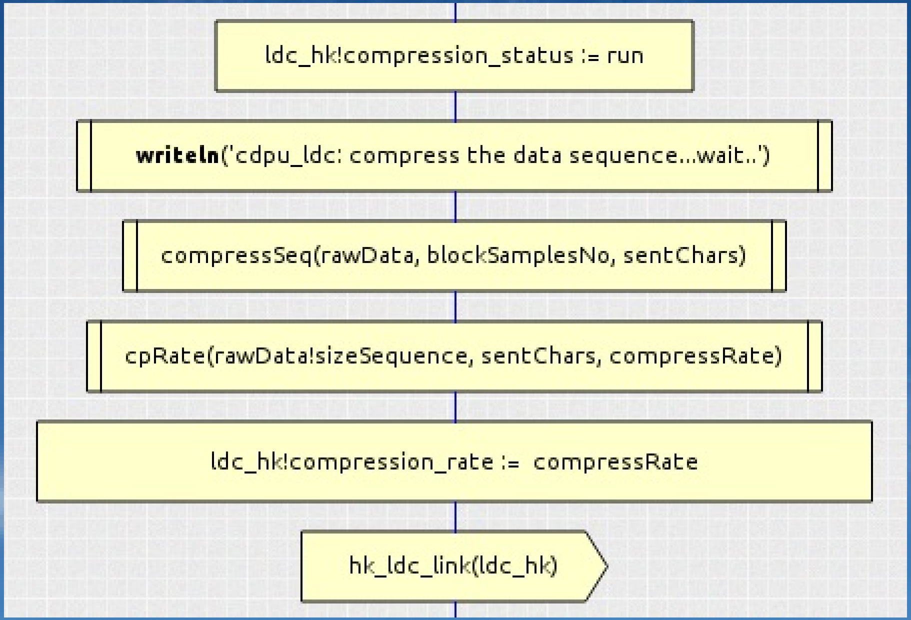
L3Obis: HK report

- *cdpu_ldc* → HK data:

compression rate & status

→ *hk_ldc_link* PI → *cdpu_tc_hk*




ldc_hk!compression_status := run
writeln('cdpu_ldc: compress the data sequence...wait..')
compressSeq(rawData, blockSamplesNo, sentChars)
cpRate(rawData!sizeSequence, sentChars, compressRate)
ldc_hk!compression_rate := compressRate
hk_ldc_link(ldc_hk)

- *compRate* function:

→ compression rate = IN/OUT

- general TM fields:

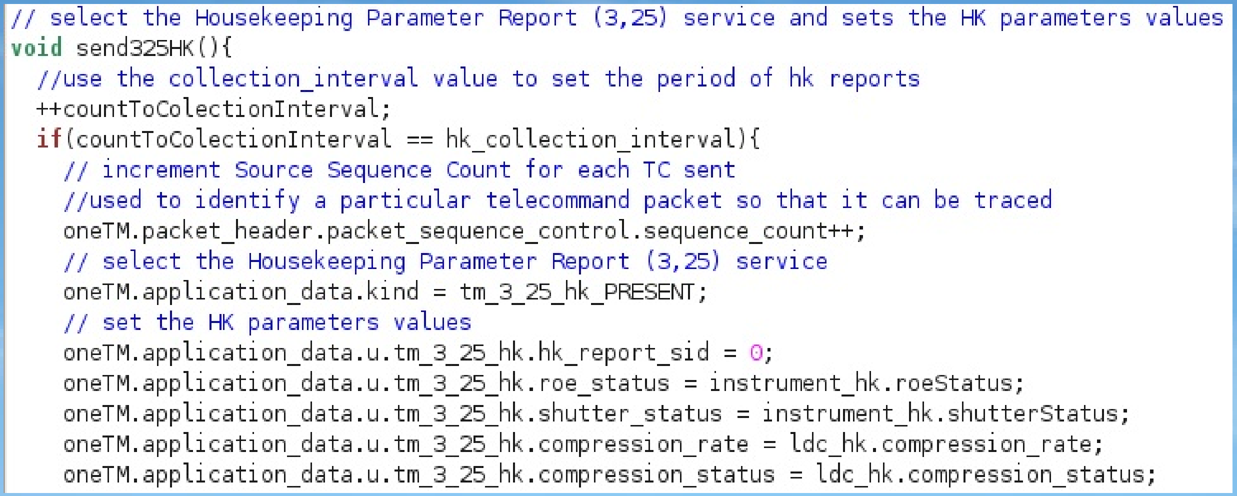
→ *cdpu_tc_hk_startup*()

- specific TM fields:

→ *send325HK*()

- *tm_link* → send TM packets

→ GUI

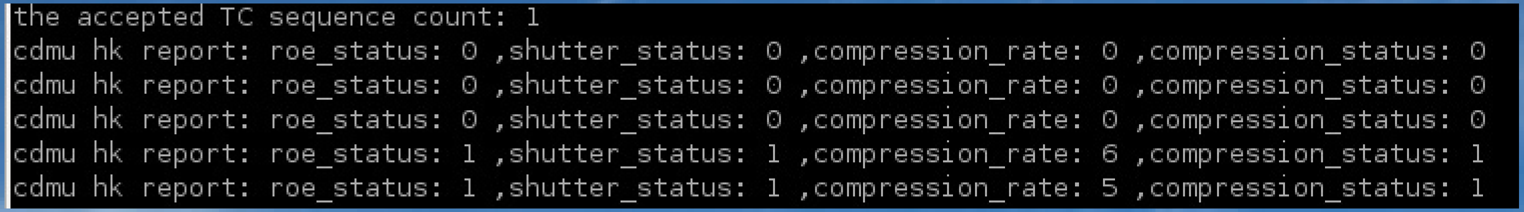


```
// select the Housekeeping Parameter Report (3,25) service and sets the HK parameters values
void send325HK(){
  //use the collection_interval value to set the period of hk reports
  ++countToColectionInterval;
  if(countToColectionInterval == hk_collection_interval){
    // increment Source Sequence Count for each TC sent
    //used to identify a particular telecommand packet so that it can be traced
    oneTM.packet_header.packet_sequence_control.sequence_count++;
    // select the Housekeeping Parameter Report (3,25) service
    oneTM.application_data.kind = tm_3_25_hk_PRESENT;
    // set the HK parameters values
    oneTM.application_data.u.tm_3_25_hk.hk_report_sid = 0;
    oneTM.application_data.u.tm_3_25_hk.roe_status = instrument_hk.roeStatus;
    oneTM.application_data.u.tm_3_25_hk.shutter_status = instrument_hk.shutterStatus;
    oneTM.application_data.u.tm_3_25_hk.compression_rate = ldc_hk.compression_rate;
    oneTM.application_data.u.tm_3_25_hk.compression_status = ldc_hk.compression_status;
```

L3Obis: HK report

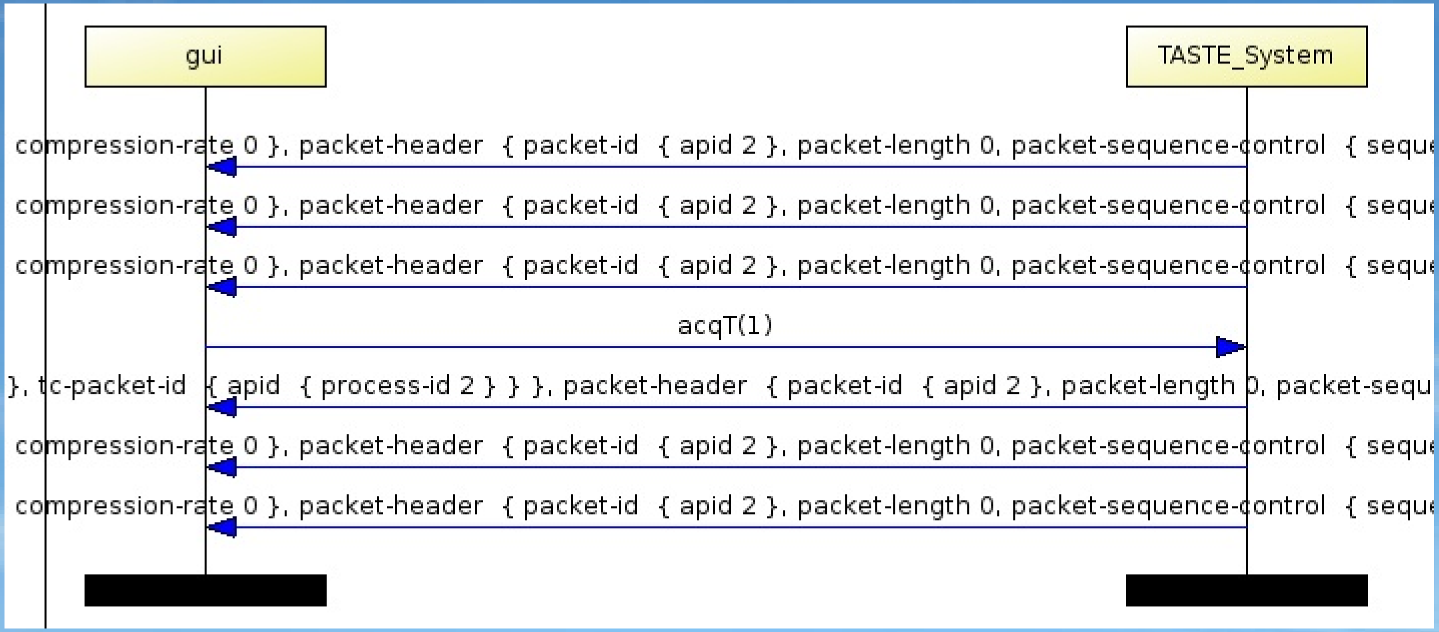
- monitor HK parameters:

→ console



```
the accepted TC sequence count: 1
cdmu hk report: roe_status: 0 ,shutter_status: 0 ,compression_rate: 0 ,compression_status: 0
cdmu hk report: roe_status: 0 ,shutter_status: 0 ,compression_rate: 0 ,compression_status: 0
cdmu hk report: roe_status: 0 ,shutter_status: 0 ,compression_rate: 0 ,compression_status: 0
cdmu hk report: roe_status: 1 ,shutter_status: 1 ,compression_rate: 6 ,compression_status: 1
cdmu hk report: roe_status: 1 ,shutter_status: 1 ,compression_rate: 5 ,compression_status: 1
```

→ MSC tool : trace in real-time the message exchanges



compression-rate 0 }, packet-header { packet-id { apid 2 }, packet-length 0, packet-sequence-control { seque

compression-rate 0 }, packet-header { packet-id { apid 2 }, packet-length 0, packet-sequence-control { seque

compression-rate 0 }, packet-header { packet-id { apid 2 }, packet-length 0, packet-sequence-control { seque

acqT(1)

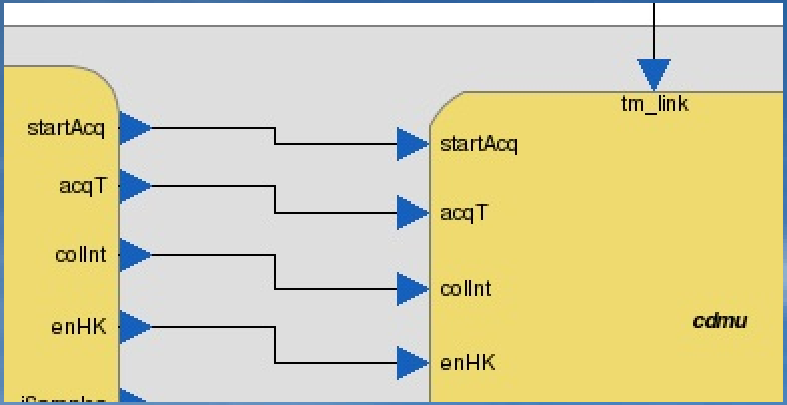
}, tc-packet-id { apid { process-id 2 } } }, packet-header { packet-id { apid 2 }, packet-length 0, packet-sequ

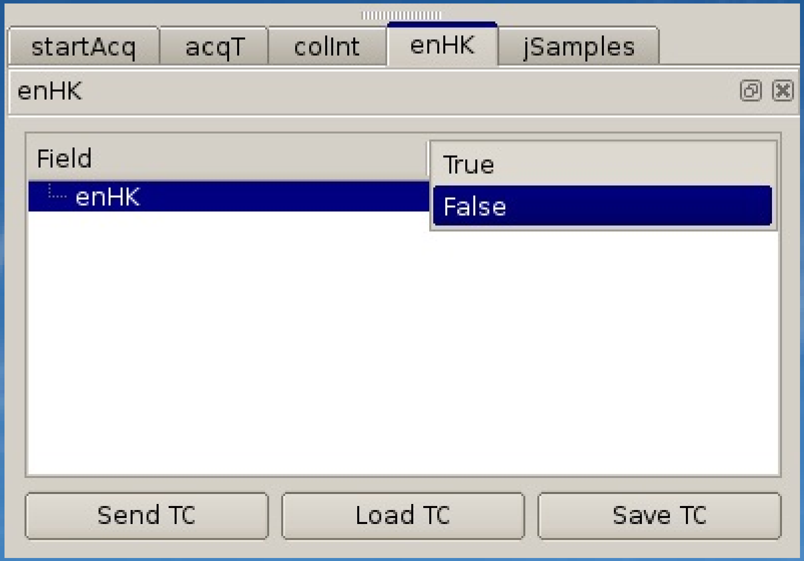
compression-rate 0 }, packet-header { packet-id { apid 2 }, packet-length 0, packet-sequence-control { seque

compression-rate 0 }, packet-header { packet-id { apid 2 }, packet-length 0, packet-sequence-control { seque

L3Obis: Enable/Disable HK Parameter Report Generation

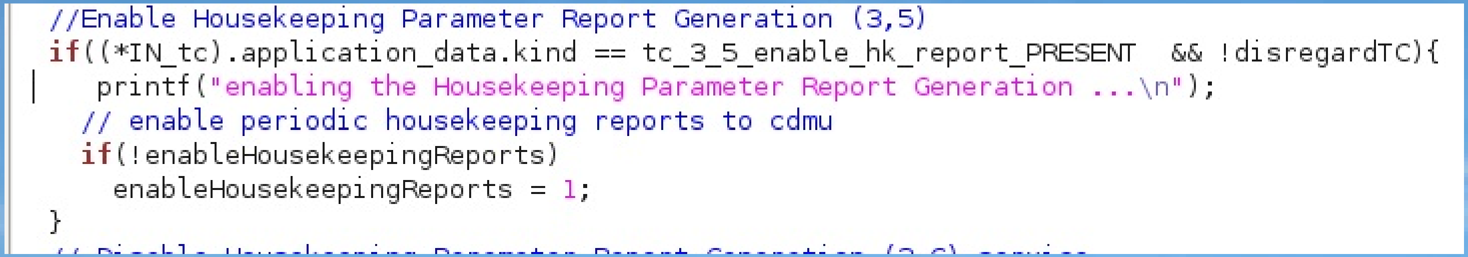
- GUI → set *enHK*





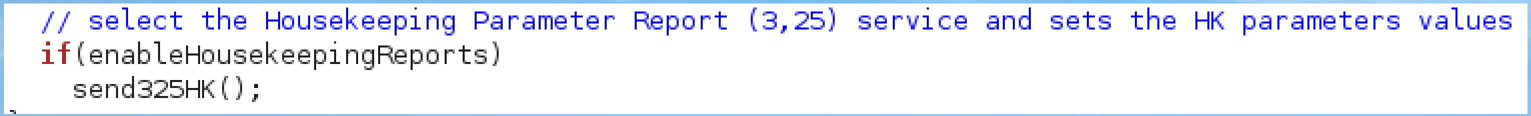
- *cdmu* → send TC of type (3,5)

- *cdpu_tc_hk*: set *enableHousekeepingReports* from TC



```
 //Enable Housekeeping Parameter Report Generation (3,5)
 if((*IN_tc).application_data.kind == tc_3_5_enable_hk_report_PRESENT  && !disregardTC){
    printf("enabling the Housekeeping Parameter Report Generation ...\n");
   // enable periodic housekeeping reports to cdmu
   if(!enableHousekeepingReports)
     enableHousekeepingReports = 1;
 }
   // Disable Housekeeping Parameter Report Generation (3,6) service
```

- check enableHousekeepingReports → send TM packets



```
 // select the Housekeeping Parameter Report (3,25) service and sets the HK parameters values
 if(enableHousekeepingReports)
   send325HK();
```

L3Obis: Future activities

- Change sample resolution:

→ 16 & 32 bits for input samples

- Test Leon2/3 performance for different input compression parameters.

- Develop the decompression software

→ recover the original uncompressed scientific data

- Fix bugs, improve code