Data Acquisition Software for the CSB_xCAL

Draft Version A

Henk Boterenbrood, January 20 1991

1 Introduction

This document describes the workings of the software developed for the T222 transputers on the **TRP**-boards controlling the CSBs (**CSB_FCAL**, **CSB_BCAL**, **CSB_RCAL**) in the **Calorimeter Second Level Trigger** and **Calorimeter Readout** systems.

The **TRP T222** transputers are connected by links in a linear chain and each T222 runs the same program. The chain of T222s gets commands and sends reports to a T800 host transputer which interacts with the VAX host computer (*CALEC*).

Messages from and to a CSB have the following protocol: **INT16 :: []BYTE**
A message always consists of:

- byte 0 and 1: source or destination transputer identifier
- byte 2: message tag
- byte 3: spare
- byte 4 and further: parameters, code or data

2 Startup of the transputer networks

Initially the **Host** T800 and **TRP T222** transputers are booted with one program code file from the host computer. All subsequent resetting and booting is carried out by the **TRP T222** and **Rebooter** under **Host**-control.

The following OCCAM code describing the start-up procedure as carried out by the **Host** is pseudo-code and is not to be taken literally; e.g. when sending a command to the CSB this should be done with an **OutputOrFail.t** routine and replies of the CSB can include error messages. This implies that there must be some separate process which handles all possible messages from the CSBs and sends on only the appropriate acknowledge messages to the procedure described below.

Before any other transputer the **Rebooter** is booted so that it can collect and store the program code of all other transputers in the system:

```
-- Reset the REBOOTER
to.csb      ! 4 :: [ REBOOTER, ARE.RESET ]
-- Wait for the acknowledge
from.csb    ? len :: [ source, ack ]
-- Boot the REBOOTER
to.rebooter ! rebooter.program.code
```

Next the **Readout** and **Trigger** transputers are booted (same is to be done for each CSB in the system):

```
-- Reset the READOUT transputers
to.csb   ! 6 :: [ BCAL, ARE.READOUT.RESET, reset.bitmask ]
-- Wait for the acknowledge
from.csb ? len :: [ source, ack ]
-- Set the broadcast select register
to.csb   ! 6 :: [ BCAL, LKC.READOUT.SBRC.SELECT, broadcast.bitmask ]
-- Send the READOUT program
send.code( to.csb, BCAL, LKC.READOUT.DATA,
           readout.program.length, readout.program )
-- Check the 'liveness' of the READOUT transputers
SEQ dest = first.readout.tp FOR no.of.readout.tp
  SEQ
    to.csb   ! 4 :: [ dest, EVT.PING ]
    -- Wait for the acknowledge
    from.csb ? len :: [ source, ack ]

-- Reset the TRIGGER transputers
to.csb   ! 6 :: [ BCAL, ARE.TRIGGER.RESET, reset.bitmask ]
-- Wait for the acknowledge
from.csb ? len :: [ source, ack ]
-- Set the broadcast select register
to.csb   ! 6 :: [ BCAL, LKC.TRIGGER.SBRC.SELECT, broadcast.bitmask ]
-- Send the TRIGGER program
send.code( to.csb, BCAL, LKC.TRIGGER.DATA,
           trigger.program.length, trigger.program )
-- Check the 'liveness' of the TRIGGER transputers
SEQ dest = first.readout.tp FOR no.of.readout.tp
  SEQ
    to.csb   ! 4 :: [ dest, EVT.PING ]
    -- Wait for the acknowledge
    from.csb ? len :: [ source, ack ]
```

Boot the **Layer2** transputers via **Trigger** transputers:

```
-- BCAL LAYER2 transputers
VAL []INT mask IS [ 1<<15, 1<<13, 1<<10, 1<<6, 1<<3 ] :
SEQ i=0 FOR 5
  SEQ
    -- Reset the LAYER2 transputer
    to.csb   ! 4 :: [ BCAL + LAYER2.TP + i, ARE.RESET ]
    from.csb ? len :: [ source, ack ]
    -- Set the broadcast select register
    to.csb   ! 6 :: [ BCAL, LKC.TRIGGER.SBRC.SELECT, mask[i] ]
    -- Send the LAYER2 program to appropriate TRIGGER
    send.code( to.csb, BCAL, LKC.TRIGGER.DATA,
               bcal.layer2.program.length[i], bcal.layer2.program[i] )

-- RCAL LAYER2 transputers
VAL []INT mask IS [ 1<<0, 1<<4 ] :
SEQ i=0 FOR 2
  SEQ
    -- Reset the LAYER2 transputer
    to.csb   ! 4 :: [ RCAL + LAYER2.TP + i, ARE.RESET ]
    from.csb ? len :: [ source, ack ]
```

```
    -- Set the broadcast select register
    to.csb   ! 6 :: [ RCAL, LKC.TRIGGER.SBRC.SELECT, mask[i] ]
    -- Send the LAYER2 program to appropriate TRIGGER
    send.code( to.csb, RCAL, LKC.TRIGGER.DATA,
               rcal.layer2.program.length, rcal.layer2.program )

-- FCAL LAYER2 transputers
VAL []INT mask IS [ 1<<0, 1<<2, 1<<5, 1<<9, 1<<12 ] :
SEQ i=0 FOR 2
  SEQ
    -- Reset the LAYER2 transputer
    to.csb   ! 4 :: [ FCAL + LAYER2.TP + i, ARE.RESET ]
    from.csb ? len :: [ source, ack ]
    -- Set the broadcast select register of FCAL CSB
    to.csb   ! 6 :: [ FCAL, LKC.TRIGGER.SBRC.SELECT, mask[i] ]
    -- Send the LAYER2 program to appropriate TRIGGER
    send.code( to.csb, FCAL, LKC.TRIGGER.DATA,
               fcal.layer2.program.length[i], fcal.layer2.program[i] )
```

Boot the **Layer3** transputers via **Trigger** and **Layer2** transputers:

```
-- LAYER3 COLLECT/ LAYER2/3 MONITOR transputer
to.csb   ! 4 :: [ RCAL + LAYER3.TP + 0, ARE.RESET ]
from.csb ? len :: [ source, ack ]
-- Set the broadcast select register of FCAL CSB
to.csb   ! 6 :: [ FCAL, LKC.TRIGGER.SBRC.SELECT, 1<<0 ]
-- Send the LAYER3 program to appropriate TRIGGER
send.code( to.csb, FCAL, LKC.TRIGGER.DATA,
           layer3.program.length[0], layer3.program[0] )

-- LAYER3 COLLECT transputer
to.csb   ! 4 :: [ RCAL + LAYER3.TP + 1, ARE.RESET ]
from.csb ? len :: [ source, ack ]
-- Set the broadcast select register of BCAL CSB
to.csb   ! 6 :: [ BCAL, LKC.TRIGGER.SBRC.SELECT, 1<<0 ]
-- Send the LAYER3 program to appropriate TRIGGER
send.code( to.csb, BCAL, LKC.TRIGGER.DATA,
           layer3.program.length[1], layer3.program[1] )
```

Boot the **Readout Collect** transputers via the **Rebooter**:

```
-- Reset BCAL READOUT COLLECT transputer
to.csb   ! 4 :: [ BCAL + RO.COLLECT.TP, ARE.RESET ]
from.csb ? len :: [ source, ack ]
-- Connect the BCAL REBOOTER link to READOUT COLLECT
to.csb   ! 4 :: [ BCAL + RO.COLLECT.TP, LKS.CONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
-- Reset RCAL READOUT COLLECT transputer
to.csb   ! 4 :: [ RCAL + RO.COLLECT.TP, ARE.RESET ]
from.csb ? len :: [ source, ack ]
-- Connect the RCAL REBOOTER link to READOUT COLLECT
to.csb   ! 4 :: [ RCAL + RO.COLLECT.TP, LKS.CONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
-- Reset FCAL READOUT COLLECT transputer
to.csb   ! 4 :: [ FCAL + RO.COLLECT.TP, ARE.RESET ]
from.csb ? len :: [ source, ack ]
```

```
-- Connect the FCAL REBOOTER link to READOUT COLLECT
to.csb   ! 4 :: [ FCAL + RO.COLLECT.TP, LKS.CONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]

-- Notify the REBOOTER so that it can
-- boot the READOUT COLLECT transputers
to.rebooter   ! notify
-- Wait for the REBOOTER to finish the booting
from.rebooter ? ack

-- The links from READOUT COLLECT transputers
-- to REBOOTER can be disconnected
to.csb   ! 4 :: [ BCAL + RO.COLLECT.TP, LKS.DISCONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
to.csb   ! 4 :: [ RCAL + RO.COLLECT.TP, LKS.DISCONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
to.csb   ! 4 :: [ FCAL + RO.COLLECT.TP, LKS.DISCONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
```

Boot the rest of the SSC transputers via the **Rebooter**:

- first subnetwork: **F/RCAL Monitor Collect** + **Monitor Host** + **F/RCAL Trigger Monitor** + **BCAL Monitor Collect**
- second subnetwork: **BCAL Data Collect** + **Data Collect**:

```
-- Reset the first subnetwork
to.csb   ! 4 :: [ FRCAL.MON.COLLECT, ARE.RESET ]
from.csb ? len :: [ source, ack ]
to.csb   ! 4 :: [ MONITOR.HOST, ARE.RESET ]
from.csb ? len :: [ source, ack ]
to.csb   ! 4 :: [ FRCAL.TRIG.MON, ARE.RESET ]
from.csb ? len :: [ source, ack ]
to.csb   ! 4 :: [ BCAL.MON.COLLECT, ARE.RESET ]
from.csb ? len :: [ source, ack ]
-- Connect the RCAL REBOOTER link to F/RCAL MONITOR COLLECT
to.csb   ! 4 :: [ FRCAL.MON.COLLECT, LKS.CONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]

-- Reset the second subnetwork
to.csb   ! 4 :: [ BCAL.DATA.COLLECT, ARE.RESET ]
from.csb ? len :: [ source, ack ]
to.csb   ! 4 :: [ DATA.COLLECT, ARE.RESET ]
from.csb ? len :: [ source, ack ]
-- Connect the BCAL REBOOTER link to BCAL DATA COLLECT
to.csb   ! 4 :: [ BCAL.DATA.COLLECT, LKS.CONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]

-- Notify the REBOOTER so that it can boot the 2 transputer networks
to.rebooter   ! notification
-- Wait for the REBOOTER to finish the booting
from.rebooter ? ack

-- The links to REBOOTER can be disconnected
to.csb   ! 4 :: [ FRCAL.MON.COLLECT, LKS.DISCONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
```

```
to.csb   ! 4 :: [ BCAL.DATA.COLLECT, LKS.DISCONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
```

NOTE: apart from the **Readout** and **Trigger** transputers the other transputers are not checked for 'liveness' in the above OCCAM code but this can easily be done and preferably right here in this stage of the startup procedure after all other transputers are booted.

Now the ADC/DSP cards can be initialized; DSP program code and constants are sent to the **Rebooter**, which sends this code and constants to the **Readout** transputers; also trigger constants are sent to the **Trigger** transputers via the **Rebooter**:

```
-- For all READOUT and TRIGGER transputers do:
-- Connect the REBOOTER link
to.csb   ! 4 :: [ destination, LKS.CONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
-- Notify the REBOOTER so that it can send code and/or constants
to.rebooter   ! notification
-- Wait for the REBOOTER to finish sending
from.rebooter ? ack
-- The link to REBOOTER can be disconnected
to.csb   ! 4 :: [ destination, LKS.DISCONNECT.REBOOTER ]
from.csb ? len :: [ source, ack ]
```

Finally the **TRP T222** transputers can switch to Data Acquisition Mode:

```
to.csb   ! 4 :: [ BCAL, END.OF.STARTUP.SEQ ]
to.csb   ! 4 :: [ RCAL, END.OF.STARTUP.SEQ ]
to.csb   ! 4 :: [ FCAL, END.OF.STARTUP.SEQ ]
-- Wait for the acknowledges
from.csb ? len :: [ source, ack ]
from.csb ? len :: [ source, ack ]
from.csb ? len :: [ source, ack ]
```

In the above OCCAM code procedure **send.code()** is defined as follows:

```
PROC send.code( CHAN OF ANY to.csb,
                VAL INT16   destination,
                VAL BYTE    message.tag,
                VAL INT     program.length,
                VAL []BYTE  program )
  INT index, bytes.to.send, mess.length :
  SEQ
    -- Send the program in chunks with ca. 32 kByte maximum size
    index         := 0
    bytes.to.send := program.length
    WHILE bytes.to.send > 0
      SEQ
        IF
          bytes.to.send > (#00007FFF - PAR.INDEX)
            mess.length := (#00007FFF - PAR.INDEX)
          TRUE
            mess.length := bytes.to.send
        bytes.to.send := bytes.to.send - mess.length
        to.csb ! INT16 mess.length :: [ destination, message.tag,
                            [program FROM index FOR mess.length] ]
        index := index + mess.length
:
```

3 Host commands to the running system

Once the startup sequence completed successfully the CSB must be set to enable the data acquisition:

```
-- Set the broadcast select registers for distribution of the GSLT
to.csb ! 6 :: [ BCAL, LKB.BRC.SELECT, bcal.broadcast.bitmask ]
to.csb ! 6 :: [ RCAL, LKB.BRC.SELECT, rcal.broadcast.bitmask ]
to.csb ! 6 :: [ FCAL, LKB.BRC.SELECT, fcal.broadcast.bitmask ]
-- 'Switch on' the links to READOUT COLLECT
SEQ link.no = 0 FOR 4
  SEQ
    to.csb ! 6 :: [ BCAL, LKS.ENABLE.READOUT.LINK, link.no ]
    to.csb ! 6 :: [ RCAL, LKS.ENABLE.READOUT.LINK, link.no ]
    to.csb ! 6 :: [ FCAL, LKS.ENABLE.READOUT.LINK, link.no ]
```

Now the system is ready for data-taking and the **Host** can send messages to the CSBs with one of the following message-tags:

- **LKB.BRC.SELECT**: set the LKB broadcast select register for distribution of the GSLT
- **LKS.ENABLE.READOUT.LINK**: switch on (enable usage of) one of the 4 links to the **Readout Collect**
- **LKS.DISABLE.READOUT.LINK**: switch off one of the 4 links to the **Readout Collect**
- **LKC.READOUT.SBRC.SELECT**: set the LKC broadcast select register for the **Readout** transputers
- **LKC.TRIGGER.SBRC.SELECT**: set the LKC broadcast select register for the **Trigger** transputers
- **LKC.READOUT.DATA**: send the following bytes (containing e.g. a Run-Control command, DSP-code or parameters) to selected **Readout** transputers via an LKC broadcast
- **LKC.TRIGGER.DATA**: send the following bytes (containing e.g. a Run-Control command or parameters for the trigger algorithm) to selected **Trigger** transputers via an LKC broadcast

4 Requests from READOUT and TRIGGER to CSB

Via their LKC-link **Trigger** and **Readout** transputers can send the following one-byte requests to the **TRP T222**:

- **LKS.CONNECT.READOUT**: request to connect me via LKS to one of the links to the **Readout Collect**
- **LKS.DISCONNECT.READOUT**: request to disconnect my connection via LKS to one of the links to the **Readout Collect**
- **LKS.CONNECT.MONITOR**: request to connect me via LKS to one of the links to the **Trigger Monitor** or **Monitor Collect**
- **LKS.DISCONNECT.MONITOR**: request to disconnect my connection via LKS to one of the links to the **Trigger Monitor** or **Monitor Collect**