Metrum Information Storage Ltd. - Wells, England. BA5 1BB

PROJECT NUMBER	1098
DATE	29/10/97

SOFTWARE NOTES FOR THE JIVE/Mark IV STATION UNIT

APPROVAL:

AUTHOR:.....P.A.Hazell

PROJECT ENGINEER: .....S.M.Parsley

CUSTOMER: .....

SU Software

SU Software Document Over view **7**

Software Build Guide **7**

Tools 7
TOOL CHAIN:P&G PC development for SUCC and Micro Boards. 7
TOOL CHAIN: P&G PC development for CCC simulation. 8
PC development file system for SUCC and Micro Boards 8
Include Files 9
Optimisation 9
The Build Process 9

SUCC Software **11**

The SUCC Context and Function. 11
The SUCC Software Scope. 12
Tasks in the system. 12
'IDLE' 13
'ROOT' 13
'PNAD' 13
'DIAG' 13
'RDQU' 13
'SRVR' 13
'D2C3' 13
'UDPT' 13
'TRMC' 14
'AUTO' 14
'CRMC' 14
'DCMC' 14
'DQDC' 14
'DDRB' 14
'TAPE' 14
'HEAD' 14
'DCOM' 14
'POLL' 14
'SVTP' 14
'ABRT' 14
'SERV' 15
'SUIC' 15
'USES' 15
SUCC Queues 15
'DIAG' 15
'DCMQ' 15
'SMCQ' 15
'TRMQ' 16

'CRMQ' 16
'PCMQ' 16
'DQCQ' 16
'SERV' 16
'D2C3' 16
'PDUR' 16
'PTCQ' 16
'PHCQ' 17
'DCCQ' 17
'ABTQ' 17
'DPUC' 17
'SUIQ' 17
'USEQ' 17
SUCC Semaphores 17
'MBSM' 17
'RQSM' 18
'QCON' 18
Example C3 Message Handling 18
C3BUFF ownership 19
ROT Reconstituted Observe Time and Command queuing 19
Diagnostics 19
Error Codes 19
SUCC Status Bits 19
SUCC Status Bit Definitions 20

SUCC Micro Board Message System 22

Message system Dual Port Ram contents. 22
Message system initialisation 22
Message System Operation 23
Additional Dual Port Ram Contents 23

TRM Firmware 24

Overview 24
Build information 24
TRM Tasks 24
'IDLE' 24
'ROOT' 24
'DIAG' 24
'pxil' 25
'XFRM' 25
'STAN' 25
'MSGH' 25
Interrupt service routines 25
"succisr" Dual Port Ram interface interrupt service routine 25
"fpgaisr" Fpga interrupt service routine 26
TRM Queues 26
TRM Semaphores 26
TRM File descriptions 27
FPGADATA.S 27
SUCCISR.S 27
FPGAISR.S 27
FPGA_CON.CC 27
PROG_XIL.CC 27
DATAMGR.CC 27

FPGASERV.CC ... 27
VLBA_XIL.CC ... 27
MKIV_XIL.CC ... 27
ROOT.CC ... 27
TRANSFER.CC ... 27
MSGHNDL.CC ... 27
Various include files ... 27
Message types defined for SUCC / TRM communication. ... 28
eProgFpga: ... 28
eConfiguration: ... 28
eTrmReqStatistics ... 28
eNextTotToTx ... 28
eFlushDeskewRam: ... 29
eTrmLoadRam: ... 29
eTrmReadRam: ... 29
eDebug: ... 29
eSendTestData: ... 29
eTrmTransmitAddress: ... 29
eRequestTapeFrameHeader ... 29
eRequestTapeFrame: ... 29
eFreeze: ... 30

CRM Firmware ... 31

Overview ... 31
Build information ... 31
CRM Tasks ... 31
'IDLE' ... 31
'ROOT' ... 31
'pxil' ... 31
'ckmx' ... 31
'ctpi' ... 31
'tabl' ... 32
'xbar' ... 32
'load' ... 32
'unrl' ... 32
'msge' ... 32
Message Handling ... 32
eCrmConfiguration ... 32
eStartTransfer ... 32
eStopTransfer ... 32
eProgFpga ... 32
eCrmLoadRam ... 32
Interrupt service routines ... 32
"succisr" ... 33
CRM Semaphore Usage ... 33
CRM Event usage ... 33
pSOS Message queue usage ... 33
CRM files ... 33

DCM Firmware Description ... 35

Overview ... 35
Build information ... 35
DCM Tasks ... 35
'IDLE' ... 35

'ROOT' 35
'FPGA' 35
'ROTC' 35
ROT_INTERRUPT 35
SYSTICK_INTERRUPT 36
eNewRot 36
eDcmSetRot 36
eDcmCheckRot 36
eDcmConfiguration 36
'TOTC' 36
SOT_INTERRUPT 36
eDcmSotTot 36
BOCF_INTERRUPT 37
eBocf_S_INSCAN : 37
eBocf_S_RUNDOWN : 37
eBocf_S_IDLE : 37
'MSGE' 37
Interrupt Service Routines 37
Dual Port Ram interface interrupt service routine 37
Fpga interrupt service routine 37
DCM / SUCC message types. 37
eDcmLoadRam: 37
eDcmReadRam: 38
eProgFpga: 38
eDcmConfiguration: 38
eDcmSetRot: 38
eDcmCheckRot: 38
eDcmTxStartRot: 38
eNewRot: 38
eDcmSotTot: 39
eBocfData: 39
eBocfReq: 39
eFreeze 39
eAbort 39
eDebug 39
eFlushDelayRam 39
eReadTotData: 39
DCM Queues 39
'ROTQ' 39
'TOTQ' 39
'DBGQ' 39
DCM Semaphores 40
'INIT' 40
'MSGQ' 40
DCM files 40
DCM Front Panel LED's 41

Workarounds 43

BOCF SYSTICK ALIGNMENT 43
SUIM / DCM setup. 43

Yourdon Design 44

SUCC Yourdon Diagrams: 44

Context diagram for the SUCC. 45
SUCC. 46
DIM CONTROL. 47
DM CONTROL. 48
PROCESS DM CONTROL. 49
PROCESS DM RESPONSE. 50
TRM CONTROL. 51
PROC TRM CONTROLS. 52
PROC TRM RESPONSES. 53
CRM CONTROL. 54
CONTROLS DISPATCHER. 55
SU CONTROLLER. 55
ROT BASED QUEUES. 56
DPU CONTROL. 57
CONTROL POS HEAD CMDS. 58
DO POSITION HEAD. 59
DO PEAK HEADS. 60
CONTROL DPU CMDS. 60
CONTROL POS TAPE CMDS. 61
POS TAPE TRANSITION DIAGRAM. 62
SERVO DPU. 63
SERVO DPU STATE TRANSISTION DIAGRAM. 64
POLL DPU. 64
DEQUEUE DPU COMMANDS. 65
DEQUEUE DPU COMMANDS STATE TRANSITION DIAGRAM. 66

SU Software Document Over view

This document is intended to be a guide for the maintenance of the SU firmware. The tool chain and build methods are given. Each major component is described from a general view point rather than in precise detail. Using this information as a guide one should be able to rebuild the firmware and make modifications. The Yourdon design diagrams at the end of this document may be viewed in conjunction with the text. The text however is not dependent on the diagrams, which have been extracted from the initial design, and show some facilities which are no longer possible.

Reference documents:-

EVN42.1 “EVN Correlator Message Formats”

EVN42.2 “EVN Correlator Message Formats Message Attributes”

EVN42.3 “EVN Correlator Message Formats Station Unit Messages”

Are required to understand many of the details. Also the original design documents for the Station Unit

Document “Software Design Specification For The Evn/Mark Iv Station Unit”

Design Specification 1685xx0500 “Design Specification For The Evn/Mark Iv Station Unit”

Hardware addressing and register bit definitions for the micro boards can be found in

Document 1685020030, “Module Software Addresses for the EVN/Mark IV Station Unit”.

Where there is a discrepancy between the original specification and the implementation, information is generally provided to explain the reason for the difference.

Software Build Guide

This section is intended as a guide to building SU software. All original design was executed on a PC platform. The intention is that there should be minimal effort required to rebuild on a UNIX based platform. However there may be some instances of case sensitivity or where paths have not been correctly set up for portability. These will be eliminated as and when found.

Tools

TOOL CHAIN:P&G PC development for SUCC and Micro Boards.

Design—Yourdon using “Select Yordon”.

Text editor e.g. brief V3.1.

Integrated Systems pSOS system V2.0E/F

[includes board support for M167. Micro boards use custom support based on E340]

pROBE debugger.

Microtec compiler CCC68K v1.2 / MCC68K v4.4 linker/asm V7.0

Debug for the MVME 167

> Integrated Systems front ended XRAY debugger MRI V2.3B over an ethernet link from PC to M167 board.

Debug for micro boards

The Back Ground Debug Mode of the MC68340 is used. Microtek International PowerScope BDM pod + Powerviews V 2.2e under Microsoft Windows for work groups 3.11.

Test harness(S/W) -- 'SUTEST' + scripts. In addition a hardware signal source is required in the form of a DPU.

TOOL CHAIN: P&G PC development for CCC simulation.

The CCC is simulated using a PC and a simple program 'SUTEST' which reads menu files and performs simple actions like :-

OPEN SOCKET

SEND DATA

RECEIVE DATA

CLOSE SOCKET

This was developed using :-

Borland C IDE V3.1 (BC31)

FTP Software inc PC/TCP OnNet Developer's ToolKit 3.0 DOS/Windows.

FTP Software inc PC/TCP Network Software Kernel for DOS.

Many other miscellaneous PC utilities have been developed under BC31 to perform data conversion and the like.

The PC/TCP libraries interface with BC31. These tools are now out of date. It is assumed that further development of 'SUTEST', if required, would be transferred to a UNIX derivate.

PC development file system for SUCC and Micro Boards.

The SUCC development has followed the basic form which pSOS system would force you down. A root directory "PSS68K" for the pSOS related files is created under which the most other files belong. Below is the pss68k tree. It shows the file structure used in the SUCC environment. A similar tree exists for the micro boards. TRM,CRM,PCM&DCM. There are some differences from the norm in that C:\SUCOM contains all common user include files for the SUCC. Also the user source files for the micro boards live below C:\1098\.

This Tree out put is cut down for clarity :-

```
C:\PSS68K              The root for the pSOS system files
+---CONFIGS            Directories under here are customised per board.
|       +---SUCC               Custom config for the SUCC.
|       |                      These are configs for the micro boards
|       +---CRM
|       |
|       +---DCM
|       |
|       +---TRM
|       |
|       \---PCM
|
+---INCLUDE            SYSTEM includes But for SUCC see c:\sucom\*.*
|
+---BSPS               Board support packages.
|       +---M167               The BSPS for the SUCC
|       |       \---SRC
|
+---BIN
|       +---MSDOS
|
+---SYS
```

```
|       +---OS
|       |       |
|       |       \---SRECORDS
|       |
|       \---LIBC
|       |
|       \---SRC
|       \---SYSCLASS
|
+---APPS                        -- Only the TFTP loader is used.
|       +---TFTP
|       +---FPSP
|       +---LOADER
|
\---WD          Working directory for the SUCC under here.
        |
        +---PAH         -- This is the main development directory for SUCC code.
        +---SUCCDOC     -- Some documentation lives in here.

C:\SUCOM                All common user include files for the SUCC.

C:\1098
+---CRM_C               CRM micro source and user include files
+---DCM_C               DCM micro source and user include files
+---PCM_C               PCM micro source and user include files
+---TRM_C               TRM micro source and user include files
```

Include Files

For the SUCC we have all common include files in the directory C:\SUCOM. Which contains all our application include files and pSOS include files which have been changed. (But check the configs and bsps for changes too.) Note that the files GLMACROS.H and GLTYPES.H form the basis of the dual ported ram interface with the micro boards and as such must be kept in step with the copies used by the micro boards.

Historically the micro board software has been developed on a different PC to that used for the SUCC for ease of running Power Views. So each micro board has its own directory under c:\1098\ all the user include files are also kept locally.

Optimisation

A Specific point to be aware of:-

> The standard pSOS system “c.opt” file which is generated to provide options to the compiler enables a number of optimisations. The math.h include file has inline assembly which is not seen by the compiler. Unfortunately the compiler then optimises away important code, in particular rounding control for the FPU. A problem which causes unpredictable results. The workaround is to use the file fp.opt and ammend the make file to use this rather than c.opt. The make file in c:\pss68k\configs\succ succ.mk has been modified from the normal distrubution to create fp.opt.

The Build Process

The method of generating SUCC code is to use the MAKEFILE located in the general source directory C:\pss68k\wd\pah. This references include files in c:\sucom\ and also the makefile in the c:\pss68k\configs\succ. To provide the necessary board support package library the makefile in c:\pss68k\bsps\m167\src needs to be run. For the SUCC this is basically a one off exercise.

The file specifications in the code are intended to be platform independent so only the DOS/UNIX changes in the makefile need to be sorted out plus text file line format. [As of 971022 some of the include files still have absolute paths which will be removed as necessary.]

In C:\pss68k\wd\pah the following results are obtained:-

C:\pss68k\wd\pah> make

Will generate a file suitable for down loading with XRAY for primitve source level debugging.

C:\pss68k\wd\pah> make ram.hex

Will generate a file suitable for down loading with tftp.

It is possible to produce a romable output but this has never been considered useful, except when building the tftp loader application which resides on the M167 boards in place of the 167'bug proms distributed with the board. By using a tftp server for storing the SUCC hex file maintenance is much simplified.

The method of generating micro board firmware is to use the MAKEFILE located in the relavent source directory eg C:\1098\TRM_C. This references local include files and also the makefile in the c:\pss68k\configs\TRM_C. To provide the necessary board support package library the makefile in c:\pss68k\bsps\TRM\src needs to be run. This is basically a one off exercise, but needs to be performed if any hardware related or start up modifications are made.

C:\1098\TRM_C> make

Will generate a file suitable for down loading with PowerViews over the BDM link. for source level debugging.

C:\1098\TRM_C> make rom.hex

Will generate a file for burning into rom. The pSOS system utility 'splitrom'is used to create separate files for each rom. There are batch files in the micro directories which automate these actions.

Note that the hardware configuration of the SU; SUCC and micro boards does not exclude the potential to extend the tftp boot to idea to down loading the code for the micro boards. In principle this only requires a rom boot loader to be in rom on the micros, but would increase the SRAM size required.

SUCC Software

The SUCC Context and Function.

The function of the Station Unit (SU) as a component in the over all system is to decode and deskew data output from a Data Playback Unit (DPU) and then to pass it on correctly timed, to the Correlator. To perform this it has two kinds of interface, signal data and control. Signal data are received from the DPU, processed and passed on to the Correlator via the Station Unit Interface Module (SUIM). Control information flows between the SU and the Correlator Control Computer (CCC or C3) and also between the SU and the DPU. The local control of the SU is performed by the Station Unit Control Computer (SUCC).

The SU interfaces with the Correlator Control Computer via an ethernet link and acts as a server to which the C3 connects to send commands. The SUCC also has an RS232 9600 baud connection to the Data Play Back Unit (DPU).

The SUCC hardware is a VME based single board computer MVME167-01A which is connected to a standard VME J1 back plane. The MVME167-01A interfaces to the ethernet and RS232 connections via an MVME 712/M transition module connected to a standard J2. The J1 back plane is physically part of a custom backplane which also provides connectivity for other 'micro' boards on a custom J2 arrangement which provides signal paths for the information to be processed. The VME interface is therefor D16/A24.

The SUCC software runs as a set of tasks under the pSOS+ real time Kernel. The low level ethernet communications are handled by pNA+.
The only resident firmware in the SUCC is a simple tftp boot loader. This can be configured during powerup to set the required file imformation.

The other boards in system are:-

DIM DATA INPUT MODULE
2 off. No processor fitted. These have 64 by 64 cross bar switches which route input signals from the DPU to the TRM cards via the custom J2 bus.

TRM TRACK RECOVERY MODULE
8 off. These boards have 68340 a micro processor which control the XILINX devices which handle the actual signals. Communication between the SUCC processor and the 68340 is via a dual ported ram. The TRM micros are tasked with the responsibility of managing the transfer of data to the CRM.

CRM CHANNEL RECOVERY MODULE
1 off. This board has the same dual port interface to the VME. Its purpose is the complement of the recorders formatter.

PCM PHASE CALIBRATION MODULE
1 off. This passively monitors the signal data as it passes between the CRM and DCM.

DCM DELAY CONTROL MODULE
1 off. This is in fact a three board module stores astronomical and transfers it to the SUIM under precisely controlled timing. The DCM contains the micro and VMW interface, two DMM's which contain the delay storage.

SUIM STATION UNIT INTERFACE MODULE
1 off. Customer supplied module which provides clocks SYSTICK and
STN_CLK.

The SUCC Software Scope.

The Scope of the SUCC software is to action commands from the C3 and perform these either immediately on receipt or at a time specified. The time base for actioning commands Reconstituted Observe Time (ROT) is maintained by the clock signals from the SUIM.

In general for each type of hardware there is a separate task which handles all actions associated with it. The exceptions are the DIM's, which are configured directly as the incoming configuration is processed by the SUCC. DPU has a suite of tasks to perform communication and control.

The ethernet interface is handled by the pNA+ package, we run three tasks that interface with pNA+. SRVR, handles the incoming packets on the socket interface, UDPT which handles the datagram input for the clock messages and D2C3 is a task which handles all responses back to the C3. When a socket is opened by the SRVR task it is shared with Task D2C3. The SUCC software design as per EVN42 assumes that the correlator control computer will open a single socket to the SUCC over which all non data gram communication will pass. Any responses to the correlator control computer are tagged as per EVN42 and sent over the single socket. There are no out going datagrams defined for the SU.

The DCMC PCMC and CRMC tasks are only started if the relevant hardware is detected. The tasks need to synchronise the messaging system with the micro board before entering the normal mode of operation. The TRMC task is always started and attempts to synchronise with each TRM board fitted in turn, but will only have a single attempt per board. The other tasks will continue to attempt to synchronise indefinitely.

For the TRMC DCMC PCMC and CRMC tasks the essential mode of operation is the same. We can illustrate this by reference to the TRM control task TRMC:-
The Root task calls the init_trm_control() which exists in the TRM_CNTL.CC file. This creates and starts the TRMC task which then vies for cpu time with all other tasks.

After any initialisations which are necessary the task enters a hand shake routine with the TRM micro processors to establish the Dual Port Ram (DPU) messaging system. Until the messaging system is synchronised the tasks do not accept any inputs from their control queues. Note that the TRM is a special case in that it handles all 8 TRMS, the other control tasks only handle a single board.
The only way to gain the TRMC attention is by sending a message to the TRM control queue TRMQ. The TRMC task sits waiting for work to do using a wait for ever on the TRMQ message queue. When something arrives the message type is checked and the processing is performed as necessary.

The DPU tasks are all started by calling init_dpu_control() from root.
The ethernet communictions tasks are started by a call to init_pna(). This is currently done prior to starting most of the other tasks, but perhaps should be done after the all others.

Tasks in the system.

The probe qt command, provides a list of all tasks in the system. Using this as a guide the purpose of each task is outlined.

```
pROBE+>qt

  Name      TID    Prio Mode Status    Susp?     Parameters              Ticks
-------------------------------------------------------------------------------
 'IDLE' -#00010000  00  2000 Running
 'ROOT' -#00020000  0F  2000   Ready   YES
 'PNAD' -#00030000  FF  2001  Evwait        EVENTS = C0000000          forever
 'DIAG' -#00140000  14  0000   Qwait         Q =  'DIAG' -#00150000  00000003
 'RDQU' -#001F0000  DC  0000   Qwait         Q =  'DQCQ' -#001B0000   forever
 'SRVR' -#00210000  50  2007  Cpwait
 'D2C3' -#00220000  50  2006   Qwait         Q =  'D2C3' -#00200000   forever
 'UDPT' -#00230000  50  2007  Cpwait
 'TRMC' -#00240000  C8  2006   Qwait         Q =  'TRMQ' -#00180000   forever
 'AUTO' -#00250000  64  0000 Wkafter                                  00000002
 'CRMC' -#00260000  C8  2006   Qwait         Q =  'CRMQ' -#00190000   forever
```

```
'DCMC' -#00270000  C8  2006   Qwait         Q =  'DCMQ' -#00160000   forever
'DQDC' -#002F0000  1E  2006   Qwait         Q =  'DPUC' -#002D0000  00000006
'DDRB' -#00300000  1E  2006   Swait        SM =  'RXC4' -#00120000   forever
'TAPE' -#00310000  1E  2006   Qwait         Q =  'PTCQ' -#00290000   forever
'HEAD' -#00320000  1E  2006   Qwait         Q =  'PHCQ' -#002A0000   forever
'DCOM' -#00330000  1E  2006   Qwait         Q =  'DCCQ' -#002B0000   forever
'POLL' -#00340000  1E  2006 Wkafter                                  00000088
'SVTP' -#00350000  1E  2006  Evwait        EVENTS = 00000010         forever
'ABRT' -#00360000  1E  2006   Qwait         Q =  'ABTQ' -#002C0000   forever
'SERV' -#00370000  C8  2006   Qwait         Q =  'SERV' -#001C0000   forever
'SUIC' -#003A0000  C8  0000   Qwait         Q =  'SUIQ' -#00380000   forever
'USES' -#003B0000  C8  0000 Wkafter                                  0000000E
```

'IDLE'

pSOS idle task.

'ROOT'

Control is passed to here after the MVME167 is initialised and pSOS+ starts. We use this as the spring board to create and run the other tasks in the system. It is also where I tend to put tests such as dual port ram tests. The general means of starting tasks is to call an init procedure in the file containing the bulk of the task code. e.g. init_dcm_control().

'PNAD'

This task is the pSOS+ network handling component. It provides the services which 'SRVR' 'UDPT' and 'D2C3' use.

'DIAG'

'DIAG'is a low priority task which waits on the 'DIAG' queue to print out virtually all console diagnostics. The only exceptions being some trivial initial diagnostics and savere error condition output before the task is run.

'RDQU'

This task services the ROT based dequeuing of commands. The file ROTQ.CC contains the rask and also entry points available to other tasks for entering commands into the rot based command queue. Semaphore 'RQSM' controls access to the ROTQ data structures. The accuracy with which ROT queued commands are timed has to be traded off against extra processing in the SUCC DCMC task and on the DCM board.

'SRVR'

This task is the first part of the tape servo system. It runs when tape servo is enabled and the DCMC task has sent it a newROT indication. Its first action is to request an update of the latest time stamp from tape via TRMC. With the value for the new ROT and the TOT plus a factor set in Miscellaneous parameters it computes a servo error which is globally available. It then sets an event SERVO_EV for the task SVTP to perform the actual servo action by computing and sending the DPU commands.

'D2C3'

The purpose of this task is to send data back to the C3. 'D2C3' sits in a wait for ever on the 'D2C3' queue. 'D2C3' is a variable length queue, the message contains a union which has a fixed type position then different content dependent on the message type. For short messages the entire message may be passed over the variable length queue, if the message is longer then a pointer to the C3BUFF containing the message is all that is passed on the queue. For messages using pointers there are two possible modes either the C3BUFF is returned to the free pool of C3BUFF's or not.

'UDPT'

Some 'clock' messages are to be broadcast on the ethernet. The UDPT task runs to handle reception of these. Dispatching is virtually identical to that of commands received using the socket interface, so provided an unreliable link is acceptable any command can in fact be broadcast! If responses are generated

by acting on such commands they will, however be directed to the currently open C3 socket using the queue 'D2C3'.

'TRMC'

This task handles all communication between the SUCC and the TRM micro boards. It waits for ever on queue 'TRMQ'. It services both C3 generated commands and requests made by other tasks in the SU.

'AUTO'

This task is intended to provide an auto stimulus which can be used for triggering a request for a header. The resulting header can be used for checking correct track position. [The code exists for this but we do not use it and it is not fully tested]. EVN 42 defines the Auxiliary Check table data against which the header is compared.

'CRMC'

All communication with the CRM micro is performed through this task.

'DCMC'

The DCMC task is one of the most busy tasks in the system. All communication with the DCM micro board is handled, which includes the tracking of ROT, computation of header data for the BOCF headers and stimulating the SUIM when a BOCF event has occurred.

'DQDC'

This task handles the dequeuing of C3 commands which require interaction with the DPU. They are passed on to the respective task for action. If there is no outstanding command the queue operation times out and sends an event to stimulate the 'POLL' task.

'DDRB'

Responses from the DPU are built by this task.

'TAPE'

Position tape commands to footage or TOT are handled by this task.

'HEAD'

Position Head and peak heads commands are processed by this task.

'DCOM'

DPU Commands - transparent commands and DPU configuration are executed by this task.

'POLL'

The POLL task is triggered by a time out in DQDC and polls the DPU. To maintain a reasonably current view of the DPU Several locations are read. Location 73 for general status, the current footage and the currently enabled heads position. To minimise latency for any other contact with the DPU only one read is performed per activation. The head position is read infrequently and is only the currently enabled head. Since the C3 can independently make head selections using direct commands the ‘POLL’ task does not change the enabled head to update all fitted head positions.

'SVTP'

Servo Tape waits on event SERVO_EV then computes the servo requirements using the current servo error as computed by task 'SERV'.

'ABRT'

This task handles the DPU related aspects of the abort. [This is an area of code which may benefit from further investigation since the use of pSOS system for serial communication implies using the semaphores

to gain access to the port. If a task is aborted with out returning the semaphore we could get a lock up condition.

'SERV'

To compute servo error two ingredients are required. The current ROT and the latest TOT time from Tape. Since the current ROT is available and valid only at the instant that the DCMC task receives the ROT tick from the DCM we align the process to this. The 'SERV' task is activated by the DCMC task at the intervals established in the SUMISC command from the C3. On activation 'SERV' requests the latest TOT from tape from the TRMC task. In addition to the two basic quantities 'SERV' also uses the SUMISC ROT TOT offset to add an additional element. The SUMISC ROT TOT offset is just a means to allow off-line shifting of the servo pending the SUCC computing the required offset its self. (The offset is dependent on data rate out of the DM and the mux ratio.) The SUMISC ROT TOT offset was intended to enable the SU to cope with large time offsets.

'SUIC'

This task uses code supplied by EVN and processes the SUIM related commands.

'USES'

This task is updates the SUIM error store every second.

SUCC Queues

The following Queues are implemented in the SUCC.

```
pROBE+>qq

  Name       QID      TQ Len    MQ Len   MQ Limit  Buf Src  Qtype Variable
-------------------------------------------------------------------------
  'DIAG' -#00150000  00000001  00000000  0000001E   Private  FIFO   Yes
  'DCMQ' -#00160000  00000001  00000000  0000000A   Private  FIFO   No
  'SMCQ' -#00170000  00000000  00000000  0000000A   Private  FIFO   No
  'TRMQ' -#00180000  00000001  00000000  00000032   Private  FIFO   No
  'CRMQ' -#00190000  00000001  00000000  0000000A   Private  FIFO   No
  'PCMQ' -#001A0000  00000000  00000000  0000000A   Private  FIFO   No
  'DQCQ' -#001B0000  00000001  00000000  00000006   Private  FIFO   No
  'SERV' -#001C0000  00000001  00000000  00000002   Private  FIFO   No
  'D2C3' -#00200000  00000001  00000000  00000014   Private  FIFO   Yes
  'DPUR' -#00280000  00000000  00000000    none    Sys-pool  FIFO   No
  'PTCQ' -#00290000  00000001  00000000    none    Sys-pool  FIFO   No
  'PHCQ' -#002A0000  00000001  00000000    none    Sys-pool  FIFO   No
  'DCCQ' -#002B0000  00000001  00000000    none    Sys-pool  FIFO   No
  'ABTQ' -#002C0000  00000001  00000000    none    Sys-pool  FIFO   No
  'DPUC' -#002D0000  00000001  00000000  00000014   Private  FIFO   Yes
  'SUIQ' -#00380000  00000001  00000000    none    Sys-pool  FIFO   No
  'USEQ' -#00390000  00000000  00000000    none    Sys-pool  FIFO   No

Sys-pool total = 00000064
Sys-pool free  = 00000002
```

'DIAG'

All Task diagnostics for output on the console port are directed through this variable length queue.

'DCMQ'

This queue provides all input to the DCM control task. All C3 commands are entered into the queue by 'proc_DCM_CMD' which is called by 'proc_c3_msg' either directly on receipt or after ROT based queuing. Interrupts from the DCM micro board are fielded by the interrupt service routine which loads into the queue a message with a type indicator of Q_ISR_MSG.

'SMCQ'

This queue provides all input to the SUIM control task. All C3 commands are entered into the queue by 'proc_SUIM_CMD' which is called by 'proc_c3_msg' either directly on receipt or after ROT based queuing.

The processing required in SUIM after a BOCF is stimulated by the DM_CNTRL task loading the BOCF notification into this queue. There is no hardware interrupt source defined on the SUIM.

'TRMQ'

This queue provides all input to the TRM control task. All C3 commands are entered into the queue by 'proc_TRM_CMD' which is called by 'proc_c3_msg' either directly on receipt or after ROT based queuing. Interrupts from the TRM micro's are also loaded into the queue with the notification of interrupt and which TRM has interrupted. A further type of message received on this queue is the request for the current TOT. Here any task (but normally either a DPU related task or servo task) can request the current TOT. The task id and an event to set on completion are passed in the queue message.

'CRMQ'

This queue provides all input to the CRM control task. All C3 commands are entered into the queue by 'proc_CRM_CMD' which is called by 'proc_c3_msg' either directly on receipt or after ROT based queuing. Interrupts from the CRM micro are also loaded into the queue. The queue also carries non C3 commands for start and stop transfer. The start transfer is normally sent by the DCM control task. [In development we do not stop the CRM transfering data at the end of scan in the SU].

'PCMQ'

This queue provides all input to the PCM control task. All C3 commands are entered into the queue by 'proc_PCM_CMD' which is called by 'proc_c3_msg' either directly on receipt or after ROT based queuing. Interrupts from the PCM micro are also loaded into the queue. Note that there is no special facility to process requests in the absence of a PCM board.

'DQCQ'

This queue provides all input to the DQDC task which dequeues ROT based commands. Essentially the DCM control task ‘DCMC’ writes notification that a new value has been written to the CurrentSRot. The DQDC task then checks the head of its queues for commands to action. If any message to be activated has an activation time less or equal to the current ROT it will be processed by 'proc_c3_msg'. If the repeat flag is set and the next activation time is with in any end ROT specified then the command is replicated and requeued in the ROT queue. By replicating the command the task that eventually services the it can return the C3BUFF and need not Know that it is a reapting command.

'SERV'

The DCMC task writes to this queue when it is time to run the tape servo mechanism. The activation repetition period is defined in SUMISC parameters. The value in SUMISC parameters should be an integral number of seconds. The DCMC task will only check to activate a servo action if the current ROT is on an integral second. Note that with the newer rjc servo mechanism there is effectively no limit to the slew period which may result from a servo call. This may result in the task SERV running several times and setting the event which stimulates actual servo code many times. A situation which should not matter since event flags are not accumulated.

'D2C3'

The D2C3 queue is used to pass information back to the C3. The queued information can be the whole message or a pointer to the message to be transmitted. Many sources may write to this queue. In particular SRVR writes to inform the task that the socket is broken.

'PDUR'

This is the DPU response queue. Written to by the DPUresponse builder and read by the DPU commands controller.

'PTCQ'

This queue carries all the position tape commands to task TAPE found in file MPT1.CC.

'PHCQ'

This queue carries Head related commands for Peak and Position. It is read by task HEAD in file MPH.CC and written by DQDC in file DPU.CC.

'DCCQ'

All DPU related C3 commands are passed to task DQDC via this queue. 'proc_c3_msg' in proccmd.cc calls the entry 'proc_DPU_msg' in file DPU.CC.

'ABTQ'

When 'proc_c3_msg' in proccmd.cc processes an abort this queue is written. Task ABRT reads the queue and runs down DPU activity.

'DPUC'

Commands to be sent to the DPU are all sent via this queue.

'SUIQ'

Commands to be handled by the SUIM control task are lodged in this queue.

'USEQ'

This queue is created by init_suim_control in the SUIM_CTL.CC file. But does not appear to be used, so should be removed.

SUCC Semaphores

```
pROBE+>qs

  Name       SMID        Count           TQ Len            Qtype
-------------------------------------------------------------------
  'RDA1' -#00040000     00000001        00000000            FIFO
  'WRA1' -#00050000     00000001        00000000            FIFO
  'RXC1' -#00060000     00000000        00000000            FIFO
  'TXC1' -#00070000     00000000        00000000            FIFO
  'RDA2' -#00080000     00000001        00000000            FIFO
  'WRA2' -#00090000     00000001        00000000            FIFO
  'RXC2' -#000A0000     00000000        00000000            FIFO
  'TXC2' -#000B0000     00000000        00000000            FIFO
  'RDA3' -#000C0000     00000001        00000000            FIFO
  'WRA3' -#000D0000     00000001        00000000            FIFO
  'RXC3' -#000E0000     00000000        00000000            FIFO
  'TXC3' -#000F0000     00000000        00000000            FIFO
  'RDA4' -#00100000     00000000        00000000            FIFO
  'WRA4' -#00110000     00000001        00000000            FIFO
  'RXC4' -#00120000     00000000        00000001            FIFO
  'TXC4' -#00130000     00000000        00000000            FIFO
  'MBSM' -#001D0000     00000001        00000000            FIFO
  'RQSM' -#001E0000     00000001        00000000            FIFO
  'QCON' -#002E0000     00000001        00000000            FIFO

'RDA1','WRA1','RXC1','TXC1'
'RDA2','WRA2','RXC2','TXC2'
'RDA3','WRA3','RXC3','TXC3'
'RDA4','WRA4','RXC4','TXC4'
```

The above are system generated for the control of serial comms devices.

The user semaphores are :-

'MBSM'

This semaphore is used to control access to the C3 command buffer allocation data structures (such as they are). Any task can request and return a buffer so to ensure the ingtegrity of the structures the semaphore is used.

'RQSM'

This semaphore is used to control access to the ROT queuing control data structures. Entries are made by any task, withdrawal is normally made by the RDQU task. The flush queue command when actioned will delete all entries

'QCON'

This controls access to the DPU commands queue.

Example C3 Message Handling

As an example we can consider the handling of a configure message from the C3 assuming that the Socket is already established. The message will contain TRM configuration information and it is to be actioned immediately.
1) The task SRVR has a wait for ever peak request on the C3 Socket. When data is available the number of bytes read should contain all the interesting header information in particular the message size. Note that the peak does not consume the data. Simple sanity checks on the message is performed including checking the size. A small or large comms storage C3BUFF is obtained and the socket is then read until the entire message is placed into the C3BUFF.
2) The message header flags are inspected and the command is either actioned immediately or is queue for action at a later time.
3) Since in the example the command is not to be queued the commands dispatcher [proc_c3_msg() in file proccmd.cc] is used to direct the command to the required handler. [In fact because the command is a configuration command it is replicated and the copy is allocated to the TRMC.] In this case the TRM control task should be used. So the proc_TRM_CMD() entry point in TRM_CNTL.CC is called. At this point the task running is still SRVR, proc_TRM_CMD() only queues a message to TRMC. The message is a standard fixed length pSOS message which contains 4 longs, see the definition for QMSG in SUSTRUS.H. One entry the message type, indicates that one of the 4 longs is a pointer to the C3BUFF command store.
4) The TRMC task is unblocked from the wait for ever on the message queue. It runs and retrieves the message, picks up the type and then proceeds to its own dispatcher proc_TRM_CONTROLS() which calls the configuration message handler.
5) The TRM configure message handler locates the relevant section of the configuration command and then actions it.
6) To complete the configuration TRMC allocates a trmjob - This is an entry in an array of structures TRMJOB. Essentially this holds all information necessary to process the action. Each of the TRM boards has an entry and it contains status information for the current action. The TRMC task fills in the job information and uses the messaging system to send down the data to the TRM micro(s). At this point the processing is partially complete. No more can be done until the micro boards respond.
7) The Micro board(s) completes its action and signals to the SUCC by posting an interrupt. At interrupt service level the SUCC reads a location in the dual ported ram to remove the physical asserted interrupt and also commits a message to the TRMQ queue with message type Q_ISR_MSG and a board ident. Each micro board in the system publishes a unique interrupt vector which makes the providing the board ident quick and simple.
8) Meanwhile the TRMC task has resumed waiting for ever on the message queue.
9) The TRMC task is now unblocked and reads the message from the queue. Message type Q_ISR_MSG causes proc_TRM_BOARD_MSG() to be used to dequeue from the TRMC-MICRO messaging system the first entry pending. The trmjobs array is searched and the initiating job is determined. The SwtchOnMsgType() routine calls the relevant handler to progress the job. If the action to do was a configure, then the transaction with that TRM is complete, so its entry in the trmjob is marked as eTrmJob_Complete. Before leaving, each TRM entry is checked for its status. If no entry is marked as active then the complete job is finished. SwtchOnMsgType() returns with a buffer status indicator which controls the deallocation of dual port ram buffer and C3BUFF.
Here we will generally release the dual port ram buffer. However if the transaction was a memory fill for example, then the same buffer may be reused to propagate the fill.

If the transaction had completed a C3 command, then the C3BUFF will be released.

C3BUFF ownership

A simplistic approach is used to control allocation and deallocation of C3BUFF's. A C3BUFF can belong to only one task. This allows any task with control of the buffer to return it to the free pool of buffers. This was principally introduced to cover the configuration case where one C3 command would be destined for many tasks. The simple brute force approach is to replicate a command buffer so that each task has its own copy and so can return it unconditionally. Replication for configuration is only performed as required and should normally only happen when the SU is not very busy.

ROT Reconstituted Observe Time and Command queuing

The SUCC implements a Command queuing mechanism which is intended to adhere to the requirements of EVN 51 “Correlator Software Queue Specification”.
In order to maintain its idea of ROT the SU must first be configured and then given a set ROT command. The SUIM board must be configured with the correct STN_CLK rate prior to setting ROT in the SU.

Diagnostics

There are two places where diagnostics may be obtained. The console port of the M167 card provides an RS232 output suitable for a dumb monitor and is suitable for human observation. Also embedded in the EVN42 messaging are a number of items of data which may be interpreted by the C3. Virtually all diagnostic prints from within tasks have been implemented using a variable length queue DIAG rather than each task using the console directly. The reason for this is that the pSOS device driver uses a set of semaphores to control access to each port. A task requiring a print to the serial device will need to wait for the semaphore, first to gain access and then until the print is complete. Such a mechanism is not good for a responsive system. So we implement the variable length queue to pass the required diagnostic to the task DIAG. DIAG is a low priority task with the sole purpose of reading the DIAG queue and out putting the content to the console port. The only prints performed out side of this system are in the root task before the mechanism is up and running. For details of the diagnostics embedded in EVN42 messages see the Error Codes section.

Error Codes

```
Some action status messages back to the C3 contain these items.
#define BADHW                 0x00000001
used in TRM for some commands and indicates that the request TRM/Track is not available.
#define SU_ERROR_DPU_COMMS    0x00000005
Used in mph.cc and mpt1.cc
#define SU_MISC_PARAMS_HEADS_INVALID  0x00000006
Used in mph.cc
```

SUCC Status Bits

The following bit definitions are set in the SU general status long word. They are returned as part of the REQUEST SU STATUS or REQUEST AVERAGE STATISTICS command responses. This long word is cleared after transmission to the C3.

```
#define uf_TCREATE            0x80000000
#define uf_TSTART             0x40000000
#define uf_TRESTART           0x20000000
#define uf_TSUSPEND           0x10000000
#define uf_TIDENT             0x08000000
#define uf_QCREATE            0x04000000
#define uf_VQCREATE           0x02000000
#define uf_QSEND              0x01000000
#define uf_VQSEND             0x00800000
#define uf_QREC               0x00400000
#define uf_VQREC              0x00200000
#define uf_QIDENT             0x00100000
#define uf_VQIDENT            0x00080000
#define uf_EVREC              0x00040000
```

```
#define uf_SMIDENT            0x00020000
#define uf_SM_V               0x00010000
#define uf_SM_P               0x00008000
#define uf_SM_C               0x00004000
#define uf_SOC_C              0x00002000
#define uf_SOC_S              0x00001000
#define uf_UDP_E              0x00000800
#define uf_QSENDSUIM          0x00000400
#define uf_QSENDROTQ          0x00000200
#define uf_QSENDCRMQ          0x00000100
#define uf_QSENDTRMQ          0x00000080
#define uf_ROTERROR           0x00000040
#define uf_TRMERROR           0x00000020
#define uf_CHKROTERROR        0x00000010
#define uf_SP08               0x00000008
#define uf_SP04               0x00000004
#define uf_SP02               0x00000002
#define uf_SP01               0x00000001
```

SUCC Status Bit Definitions

This set of error indicator bits would not be expected in the final system. In general these indicate a type of failure rather than a particular instance. Additional observation would need to be applied to determine exactly where the error lies.

`uf_TCREATE`
A task creation error.
`uf_TSTART`
A task start error.
`uf_TRESTART`
A task restart error.
`uf_TSUSPEND`
A task suspend error.
`uf_TIDENT`
A task ident call error.
`uf_QCREATE`
A fixed length queue creation error.
`uf_VQCREATE`
A variable length queue creation error.
`uf_QSEND`
A fixed length queue send error.
`uf_VQSEND`
A variable length queue send error.
`uf_QREC`
A fixed length queue receive error.
`uf_VQREC`
A variable length queue receive error.
`uf_QIDENT`
A fixed length queue ident call error.
`uf_VQIDENT`
A variable length queue ident call error.
`uf_EVREC`
An event recive error.
`uf_SMIDENT`
A semaphore ident call error
`uf_SM_V`
A semaphore release error
`uf_SM_P`
A semaphore acquire error
`uf_SM_C`
A semaphore create error
These error bits may be difficult to see from the C3.
`uf_SOC_C`
`uf_SOC_S`
`uf_UDP_E`
They are internal network error indicators
`uf_QSENDSUIM`

This bit indicates that the DCMC task could not queue a message to the SUIM control task. It would indicate that the queue was full.

`uf_QSENDROTQ`

If the DCMC task can not send a new ROT indicator to the rot queuing dequeue task this bit is set. A severe error condition.

`uf_QSENDCRMQ`

This bit indicates that a task could not send a message to the CRMQ. A severe error condition.

`uf_QSENDTRMQ`

This bit indicates that a task could not send a message to the CRMQ. A severe error condition.

`uf_ROTERROR`

This bit is set if the fractional ROT is not a multiple of 2000000. (All 'normal' rot tick increments will be multiples of 2000000. Very occasionally if the SYSTICK from the SUIM is not precisely where expected a remainder value is added to the rot. The implication is that the SU and SUIM are no longer guaranteed to be in step.)

`uf_TRMERROR`

If a TRM micro board has more than (currently) 3 outstanding actions this bit is set. The assumption would be that the board has stopped responding.

`uf_CHKROTERROR`

This is set if there have been more "Check ROT fails" than the setting sent in the set `SU Error Level.`

`uf_SP08 uf_SP04 uf_SP02 uf_SP01`

Spare bits.

Indicator bits for su_gen_state

```
#define su_80000000           0x80000000
#define su_40000000           0x40000000
#define su_20000000           0x20000000
#define su_10000000           0x10000000
#define su_08000000           0x08000000
#define su_04000000           0x04000000
#define su_02000000           0x02000000
#define su_01000000           0x01000000
#define su_00800000           0x00800000
#define su_00400000           0x00400000
#define su_00200000           0x00200000
#define su_00100000           0x00100000
#define su_00080000           0x00080000
#define su_00040000           0x00040000
#define su_00020000           0x00020000
#define su_00010000           0x00010000
#define su_00008000           0x00008000
#define su_00004000           0x00004000
#define su_00002000           0x00002000
#define su_00001000           0x00001000
#define su_00000800           0x00000800
#define su_00000400           0x00000400
#define su_00000200           0x00000200
#define su_00000100           0x00000100
#define su_00000080           0x00000080
#define su_00000040           0x00000040
#define su_00000020           0x00000020
#define su_00000010           0x00000010
#define su_00000008           0x00000008
#define su_00000004           0x00000004
#define su_00000002           0x00000002
#define su_TAPE_SERVO_ENABLED 0x00000001
```

The only bit currently used indicates the servo enable condition. This long word is not cleared after sending to the C3.

SUCC Micro Board Message System

Message system Dual Port Ram contents.

In the following table the addresses shown are relative to the relevant Base Address. The first section of the Dual Port Ram (DPR) contents illustrated is common for all SUCC / Micro board interfaces. Each Application does have its own application specific storage which follows the common message storage.

Hex	0000	0004	0008	000C
0000	MicroRdy	SuccRdy	SuccGetPtr	SuccPutPtr
0010	Envelopes	MaxEnvelopes	DataPackets	MaxDataPackets
0020	MicroGetPtr	MicroPutPtr	FreeEnvelopeList	FreeDataPKTList
0030	Env_0_DATA	Env_0_Link	Env_1_DATA	Env_1_Link
0040	Env_2_DATA	Env_2_Link	Env_3_DATA	Env_3_Link
0050	Env_4_DATA	Env_4_Link	Env_5_DATA	Env_5_Link
0060	Env_6_DATA	Env_6_Link	Env_7_DATA	Env_7_Link
0070	Env_8_DATA	Env_8_Link	Env_9_DATA	Env_9_Link
0080	Env_A_DATA	Env_A_Link	Env_B_DATA	Env_B_Link
0090	Datapacket_0			
0190	Datapacket_1			
...				
0A90	Datapacket_9			
0B90	Start of App store			

There are a total of 10 data packet buffers and 12 envelopes. The original thought was that it may have been useful to associate more than one data packet with a particular message envelope. However this is not the case. Any operation which would require many data packets is performed by reuse of a single packet. Processing overhead is not a problem since the only users of large ammounts of data either have application specific storage defined in Dual Port Ram or are not used in normal operations.

An Envelope is a structure which contains Data and Link entries.
Env_n_Link contains an index to the next free entry in a free Envelope list.
Env_n_DATA contains an index of an associated Data Packet.

A data packet is a structure containing enum MSGTYPE, ULONG job serial number and a MESSAGE union. One component of the message union is a ULONG Link. The link is used to control the free list of data packets. Note that the Link is only valid when the data packet is unallocated. Once allocated it is overwritten with other message components, when restored to the free list it is loaded with the existing FreeDataPKTList value and the current FreeDataPKTList is updated with the index to the returned data packet.
Note that the terminology in the code indicates that the Free list entries are pointers which was the case during the first pass of development where the message system was simulated in a single processor, however it was soon realised that this was not practical when using the real dual port memory. The physical and logical addresses of the Dual Port Ram were different for the each side of the DPR. Further it was reasonably simple to used a simple index for each and treat the storage as arrays.

Message system initialisation

The initialisation of the message system is (or should be) performed by the SUCC after the system is synchronised. To synchronise the link the SUCC and Micro boards enter a dialogue via the entries labelled

MicroRdy and SuccRdy, each has to see the other change its condition before successful completion. See code for details.
The storage is initialised as follows. For the array of Envelopes each Env_n_Link contains the index of the next free in the list. The last in the list contains an end of list marker (-1). The entry FreeEnvelopeList contains the index of the first free Envelope.
For the array of Data Packets the Link entry is set to the index of the next free Data Packets. The last in the list contains an end of list marker (-1).

Message System Operation

All allocation and return of Envelopes and Data Packets is performed by the SUCC. The micro boards can only act on messages received. This eliminates any need for semaphore like operations. In general this is exactly how a master/slave situation continues. However there are exceptions in the case of the SUCC/DCM where message buffers are sent down to the DCM in anticipation of future actions. These are:-

- An empty packet for the next NewRot message.
- A set of packets to use as the BOCF header request software Pipeline.
- A Check Rot message.

Since each side of the communications has its own get and put pointers (index values), linking in a new Envelope can be performed reasonable safely by setting up the new entry prior to changing the end of list indicator.
Notification of new messages to the other side only requires a simple interrupt to start a search from the current get pointer. Messages can then be unlinked in order until an Envelope with an end of list indicator is encountered. Multiple notification or notification after an enter has actually been picked up (a situation which could occur if a previous search is still in progress) should always be benign. (We shall temporarily ignore the possibility of non atomic changes from end of list (-1) which possibly could give a 0x0000FFFF or 0xFFFF0000 value ? This has not yet caused a noticeable problem, but it would be tidy to eliminate the possibility. The effort required is reasonable since both the SUCC and all the micro firmware would have to be changed at the same time.

Additional Dual Port Ram Contents

Each board type has its own application specific Dual Port Ram Contents which are define in the file GLTYPES.H.

TRM Firmware

Overview

The TRM firmware is described first in overview and then in greater detail using the pROBE printout as a guide.
The TRM as a whole has the function of deskewing the data from tape and managing its transfer to the CRM. The bulk of the work is performed by the reconfigurable Xilinx devices which manage the data reception into and transmission from the deskew ram. The data from tape is a continuos flow which is correctly framed and placed in consecutive pages in the deskew ram. Each page contains a frame. The data is later extracted on a frame by frame basis at a net rate equivalent to the input rate. At a higher system level the SUCC servo's the Data Play back Unit so that the time stamps of data input are ahead of the required data for extraction. The deskew ram contains 128 pages and so can buffer a large amount of data which reduces the need for precise servoing. The TRM firmware actions requests from the SUCC as listed below in the messages section. Operationally the SUCC requests tape header information from which the time stamp data may be extracted for servoing and other data integrity checks. Once given suitable configuration information and a time stamp reference to transmit to the CRM the firmware is required to oversee the frame data extraction to the CRM to ensure that the data transferred is of the correct time stamp. Data which is not correctly timed is marked as invalid. Assuming perfect data the only action the transfer management firmware would need to do is to ensure the data transfer starts correctly. However the data may contain discontinuities which require intervention. Another requirement is to provide error statistics information. These are gathered under interrupt and made available to the SUCC via dual port ram.

Build information

Building the firmware is via the makefile located in the trm_c directory. Refer to the BUILD document for full details of the tools and directories used.

TRM Tasks

The following is a pROBE query tasks, the out put shows a snap shot the after the TRM has been running individual task descriptions follow.

pROBE+>qt

```
  Name      TID    Prio Mode Status    Susp?     Parameters              Ticks
-------------------------------------------------------------------------------
 'IDLE' -#00010000  00  2000 Running
 'ROOT' -#00020000  14  2000   Ready   YES
 'DIAG' -#00070000  14  0000   Qwait         Q =  'DIAG' -#00080000  00000009
 'pxil' -#000A0000  DC  2005   Ready   YES
 'XFRM' -#000B0000  5A  2004 Wkafter                                 00000008
 'STAN' -#000C0000  14  2004 Wkafter                                 0000003A
 'MSGH' -#000D0000  64  2004  Evwait        EVENTS = 00000001         forever
```

'IDLE'

The standard pSOS idle task.

'ROOT'

This contains the root task to which control is passed after pSOS initialisation. It starts the pSOS timer and serial devices then the other tasks in the system. After performng these functions it lowers its priority and suspends.

'DIAG'

The task 'DIAG' is a low priority task used to perform the print function for console output, its only function is to wait on queue 'DIAG' for information to print. pSOS system provides serial port drivers which are

used for console output. Access is controlled transparrently to the user by semaphores. To eliminate any contention for the semaphore 'WRA1' which controls writing on the console port task 'DIAG' is used to perform all console diagnostics. This method also decouples the diagnostic print from a task which issues diagnostics. (Normally while a task performs a print using the pSOS system provided mechanism it is blocked.) Tasks send diagnostic information to the variable length queue 'DIAG'. One feature of this is that during data transfer to the CRM when task 'XFRM' is running with out any free processor time, diagnostic out put appears to stop.

'pxil'

The task programmes/configures the Xilinx devices. Programming is always performed in parallel so that all signal path Xilinx contain the same programme/configuration. In common with all micro board firmware, when the task is started it copies a default Xilinx configuration to SRAM then programmes from SRAM. On completion the task suspends and may be resumed to reprogram. We always reconfigure/programme on each configure message according to MkIV / VLBA requirements.

'XFRM'

The Xilinx fpga provides the function of correctly framing track frames from tape and placing complete frames in pages in the Deskew Ram. Hardware pointers are accessible for the current receive and transmit pages. The 'XFRM' task manages data transfer from the deskew ram to the CRM by keeping track of the Track Frame transmissions from a given starting TOT and adjusting the TX page pointer so that correctly timed data is transmitted. At the start of a scan all data is flagged invalid until the correct page pointer is determined and loaded.
One function of the fpga interrupt service routine is to capture the Track Frame Transmission Event (TFTE) and increment a store of TFTE's. Two functions provide access to the TFTE's. One gives the number of pending TFTE's and the other gives the number of pending TFTE's and then clears the count. By using these functions 'XFRM' can determine when a track frame has just started and so what the next out going time stamp should be. The deskew ram content and hence the time stamps are accessible to the micro via the Xilinx. This allows 'XFRM' either to verify correct operation or to start a track which is currently marked bad.

'STAN'

This is the stand alone statistics task. It is activated on a 5 second wake after. The flag StandaloneStatsEnable is used to ensure that it will only change the statistics bank and print out results if a scan is not in progress. The statistics themselves are captured by the fpga interrupt service routine which increments the various counts in dual port memory for the currently active bank. (when a scan is started the StandaloneStatsEnable flag is cleared. Thus the SUCC can have complete control over the statistics bank that the ISR uses.)

'MSGH'

The 'MSGH' task handles the dual port ram messaging system for communications with the SUCC. See the message list defined below.

Interrupt service routines

"succisr" Dual Port Ram interface interrupt service routine

The communication between the SUCC and the TRM is via a Dual Port Ram, physically implemented in the IDT7025 Dual Port Ram chip. The device offers all the necessary electronics for access from each side plus features which simplify system operation. There are special addresses at the top of the Dual Port Ram which, when written to, can be used to generate an interrupt for the 'other side'. Reading the corresponding address clears the interrupt. (See the manufactures device description for detailed information). The Dual Port Ram interface interrupt service routine reads the address to clear the interrupt and then sets an event for the 'MSGH' task.

"fpgaisr" Fpga interrupt service routine

For each of the eight Xilinx fpga, all interrupt sources set bits in one internal status register. There is a single interrupt line out of each Xilinx which is used as an input to a simple PLD which asserts one IRQ line into the processor. The Fpga interrupt service routine basically picks out the Track Frame Transmit Event from a selected Xilinx and the RX frame information for each track. The TFTE is asynchronous with RX frame. Due to skew in the arrival of data the RX frame information is not guaranteed to be available for all tracks at the same time. However the RX data skew is small and a small pure delay before servicing each Xilinx for RX frame information appears to be sufficient to ensure that data from all active tracks is ready. RX frame service is basically to increment counts of error conditions dependent on the bits set in the status registers. Parity error counts are added for the frame are added only if the frame has not been rejected as having to many parity errors.

TRM Queues

```
pROBE+>qq

  Name       QID      TQ Len    MQ Len   MQ Limit  Buf Src  Qtype Variable
-------------------------------------------------------------------------
  'DIAG' -#00080000  00000001  00000000  0000001E   Private  FIFO   Yes

Sys-pool total = 00000064
Sys-pool free  = 00000064
```

The only message queue used is 'DIAG' which carries text diagnostics for output on the console port, see the description for the 'DIAG' task.

TRM Semaphores

```
pROBE+>qs

  Name       SMID        Count           TQ Len            Qtype
-------------------------------------------------------------------
  'RDA1' -#00030000     00000001        00000000           FIFO
  'WRA1' -#00040000     00000001        00000000           FIFO
  'RXC1' -#00050000     00000000        00000000           FIFO
  'TXC1' -#00060000     00000000        00000000           FIFO
  'init' -#00090000     00000001        00000000           Prio
```

The system semaphores 'RDA1' 'WRA1' 'RXC1' and 'TXC1' control access to the console serial port. 'init' is used to control access to the Xilinx devices around programming to ensure that they are always programmed before any tasks use them.

```
pROBE+>qo

Resident Objects:      Name      ID         Type       Access
-------------------------------------------------------------
                      'RN#0' -#00000000     REGION      Local
                      'IDLE' -#00010000     TASK        Local
                      'ROOT' -#00020000     TASK        Local
                      'RDA1' -#00030000     SEMAPHORE   Local
                      'WRA1' -#00040000     SEMAPHORE   Local
                      'RXC1' -#00050000     SEMAPHORE   Local
                      'TXC1' -#00060000     SEMAPHORE   Local
                      'DIAG' -#00070000     TASK        Local
                      'DIAG' -#00080000     QUEUE       Local
                      'init' -#00090000     SEMAPHORE   Local
                      'pxil' -#000A0000     TASK        Local
                      'XFRM' -#000B0000     TASK        Local
                      'STAN' -#000C0000     TASK        Local
                      'MSGH' -#000D0000     TASK        Local
```

TRM File descriptions

FPGADATA.S

This provides a fixed location in SRAM to transfer Xilinx configuration data to prior to programming the Xilinx.

SUCCISR.S

This is the dual port ram / messaging system interrupt service routine. It sets an event for the 'MSGH' task when the SUCC has written to location 0x3FFC.

FPGAISR.S

This provides the interrupt service function for the fpga.

FPGA_CON.CC

Code to fill fpga registers after the initial power up programming of the fpga's. It is used only at that time and is basically redundant.

PROG_XIL.CC

Contains a task which programmes/configures the Xilinx devices. Programming is always performed in parallel so that all signal path Xilinx contain the same configuration. In common with all micro board firmware when the task is run it copies a default Xilinx configuration to SRAM then programmes from SRAM. On completion the task suspends and may be resumed to reprogramme. We always reconfigure/programme on each configure message according to MKIV / VLBA requirements.

DATAMGR.CC

This file contains many routines which may be called to access and manipulate data. Some entries are out of date and may be tidied up.

FPGASERV.CC

The stand alone statistics task resides here.

VLBA_XIL.CC

A file which is only a repository for the VLBA Xilinx configuration held in VLBA_XIL.H.

MKIV_XIL.CC

A file which is only a repository for the MKIV Xilinx configuration held in MKIV_XIL.H.

ROOT.CC

This contains the root task to which control is passed after pSOS initialisation. There are also some diagnostic and miscellaneous routines.

TRANSFER.CC

Contains the task which runs to manage data transfer from the deskew ram to the CRM.

MSGHNDL.CC

The Dual Port Ram interface messaging system is run by the 'MGSH' task in this module.

Various include files

CFHEADER.H
HARDWARE.H
TRM.H
QMSG.H
SYS_CONF.H

SHOWBITS.H
GLTYPES.H
GLMACROS.H
VWSTRUCT.H
VLBA_XIL.H
MKIV_XIL.H
XILHEADR.H

Message types defined for SUCC / TRM communication.

eProgFpga:

This was intended to provide a means of reprogramming the Xilinx devices with a configuration sent down via memory loads into SRAM. In principle this is still possible but the configuration message below will just reconfigure with a MKIV or VLBI from read only memory!

eConfiguration:

The required configuration is checked for MKIV or VLBI and the necessary XILINX code is loaded into SRAM and the 'pxil' task is resumed. To prevent the 'XFRM' task accessing the fpga during reconfiguration the TransferEnable flag is cleared. Also the StandaloneStatsEnable flag is cleared.

It has been found necessary to perform the following sequence prior to
reprogramming the Xilinx devices.

Disable Rx into deskew ram.
Remove the Xilinx interrupt enable in the Xilinx control register.
Read the Xilinx status registers.
Delay for 10 pSOS ticks.
DisableIRQ5.
Delay for 10 pSOS ticks.

We have found that there could be unpredictable results if the above was not done. The assumption was that the processor could be adversely effected if the IRQ line had a glitch.

After reprogramming of the Xilinx is complete the new application configuration is loaded into the Xilinx control resisters. The clock recovery register is loaded as required. The StandaloneStatsEnable flag is set and the active tracks are marked in the statistics reporting areas of the Dual Port Ram.

NOTE: Before the configuration message is sent down from the SUCC the number of Track Frames per second of TOT must be loaded into Dual port Ram. The track frames per second are used to build an array which contains the fractional seconds of the Time Stamps.

A switch is performed on the PCB issue:-

Issue 0 : No action. (After Xilinx programming the MC68340 counter timers are setup to provide clocking for the front panel LED array.)
Issue 1 : The active tracks are set in the additional Xilinx to control the led display.

eTrmReqStatistics

This is now redundant. All statistics are gathered directly into Dual port Ram by the fpga intrrupt service routine. The SUCC extracts the information and controls the active statistics bank directly.

eNextTotToTx

SetNewTXTot() loads the next TOT to transmit tot counter. The CRM should not be requesting Track Frames at this time but there may be a count of TFTE's which are cleared by a call to GetTrackFrames().

All Xilinx are loaded and primed with a nominal page number to transmit. The TXC_FORCE_INVALID bit is asserted for all tracks. The TxStatus array is loaded with the status for each track. Unused tracks are marked with X and all others are marked B for Bad. The TransferEnable flag is set true and the StandaloneStatsEnable is set false.

Function TSprint(pSuccCmd->MsgBody.NextTotToTx.Tot) is called to out put a diagnostic showing the required TOT.

eFlushDeskewRam:

eTrmLoadRam:

eTrmReadRam:

eDebug:

These are still available but unused.

eSendTestData:

eTrmTransmitAddress:

These are now not available. They were initially provided to use the TRM as an electronic data source. But the Xilinx configurations now do not support this function.

eRequestTapeFrameHeader

The SUCC may request a header for either a specific track or from any track. In both cases the function QGetLastGoodHeader() is used. But the number of frames permitted in the search is different. For a request from a specific track QGetLastGoodHeader() is instructed to start at the last frame placed in a Deskew ram page and work back up to 10 frames to find a good frame. When any track is specified QGetLastGoodHeader() will only work back 4 frames. This is to minimise the time taken in the case of poor data on all tracks, since the search will continue in turn to each active track on the TRM

eRequestTapeFrame:

This was developed mainly for test purposes and should only be used after an eFreeze has been requested. There are four possible variants of this request:-
Get Tape Frame Data Any Track Given Time
Get Tape Frame Data Given Track Given Time
Get Tape Frame Data Any Track Any Time
Get Tape Frame Data Given Track Any Time

Separate functions are used for each. The result mechanism is the same for all functions. If a frame is found its contents are copied to the Dual port ram and the TrackNumber is loaded into the SUCC request message to indicate success. Otherwise the TrackNumber is set to -1.

void GetTapeFrameDataAnyTrackGivenTime(TAPEFRAMEDATA * ptr)
All active tracks will be searched using SearchForTotInDeskew() until a frame with the correct Time stamp is found.

void GetTapeFrameDataGivenTrackGivenTime(TAPEFRAMEDATA * ptr)
A Given track is searched using SearchForTotInDeskew().

void GetTapeFrameDataAnyTrackAnyTime(TAPEFRAMEDATA * ptr)

This uses the TXStatus to get tracks which are not 'X' to check. It will check as many tracks as available until a good time is found by GetLastGoodTot().

void GetTapeFrameDataGivenTrackAnyTime(TAPEFRAMEDATA * ptr)
GetLastGoodTot() is used to search the entire deskew ram for the given track for a good Time Stamp.

eFreeze:

The purpose of this command is to stop further data input. Thus leaving the content of the deskew rams available for inspection. There is no effect on output.

CRM Firmware

Overview

The CRM has only a limited software roll. Xilinx configuration, control register loading, Xbar switch loading (at the frame rate for debarrel roll) and starting and stopping data transfer between the TRM's and the DCM/DMM's. These actions are initiated by the SUCC sendiing commands via the dual port ram messageing system which is the same as in the TRM,PCM and DCM.

In general all user firmware is written in C. There is one assembly language routine which is used to handle the SUCC/Micro messaging interrupt. (This is not very time critical and could be written as a C interrupt routine.)

Build information

Building the firmware is via the makefile located in the trm_c directory. Refer to the BUILD document for full details of the tools and directories used.

CRM Tasks

The following is a pROBE query tasks, the out put shows a snap shot the after the CRM has been running individual task descriptions follow.

```
pROBE+>qt

  Name      TID    Prio Mode Status    Susp?     Parameters              Ticks
-------------------------------------------------------------------------------
 'IDLE' -#00010000  00  2000   Ready
 'ROOT' -#00020000  05  2000   Ready
 'pxil' -#00080000  DC  2005   Ready   YES
 'ckmx' -#00090000  D2  2005   Ready   YES
 'ctpi' -#000A0000  C8  2005   Ready   YES
 'tabl' -#000B0000  BE  2005   Ready   YES
 'xbar' -#000C0000  B4  2005   Ready   YES
 'load' -#000D0000  AA  2005   Ready   YES
 'msge' -#000E0000  A0  2004  Evwait        EVENTS = 00000001         forever
 'unrl' -#000F0000  96  2000 Running
```

'IDLE'

pSOS idle task, this should never run under normal conditions.

'ROOT'

The first user task which starts the pSOS tick timer, the serial driver and brings up the other tasks after which it lowers its priority.

'pxil'

This is run as part of the power up to program the xlinx devices. After the first configuration it suspends and is resumed on each configuration.

The following group of tasks perform simple operations as part of power up or reconfiguration. In general they will wait for the 'init' semaphore before performing their actions. This is to ensure that no writing can be performed to Xilinx registers befor the Xilinx has been programmed.

'ckmx'

This loads the clock selection and Mux control registers.

'ctpi'

This loads the cross track parity registers(active but not required).

'tabl'

Only used at powerup to load a default 1:1 Xbar table.

'xbar'

Transfers the Private Xbar table Phase zero to the Xbar holding regs.

'load'

Transfers the Xbar holding reg information to the switches.

'unrl'

This task loops inspecting the state of a bit indicating the end of a track frame. At the end of a track frame it increments the count for the current phase of barrel roll and changes phase if required. The Cross Bar Switch is always loaded at the end of a track frame. This arrangement loads the processor at near worst case irrespective of the barrel roll state. Apart from the root task this is the lowset priority user task in the system. This low priority protects it from causing bus errors during Xilinx reprogramming.

'msge'

This Task synchronises the message system with the SUCC. Once the message system is running it waits on an event to be activated. The messageing system is the same as in the TRM,PCM and DCM. See Message Handling below for messages actioned.

Message Handling

Task 'msge' has provision for the following message types.

eCrmConfiguration

On receipt of the Configuration message the required Xilinx configuration is determined by inspection of the mux ratio. The Xilinx configuration is copied into temporary store and the Xilinx devices are reprogrammed by resuming task 'pxil'. When programming is complete the cross track parity, mux control and clock control registers are loaded. Also the first phase of the barrel roll sequence is loaded into the cross bar switch by restarting xbar then load. Note that the Cross Bar Table data needs to be transformed before loading into the switch. (Odd and even bytes of the data word are set to the same value.) This prevents the SUCC from changing the Barrel roll configuration on the fly.

eStartTransfer

This simply manipulates the necessary control bits to start off the transfer.

eStopTransfer

This simply manipulates the necessary control bits to stop the transfer.

eProgFpga

See eCrmLoadRam.

eCrmLoadRam

These messages are still available but are of no value. The reason is that now each time the CRM is configured, the Xilinx devices are reprogrammed and the new configuration is loaded. There is no hardware restriction, only an additional layer of software control.

Interrupt service routines

There is one assembly language routine which is used to handle the SUCC/Micro messaging interrupt. This is not very time critical and could be written as a C interrupt routine.

"succisr"

Dual Port Ram interface interrupt service routine. The communication between the SUCC and the CRM is via a Dual Port Ram physically implemented in the IDT7025 Dual Port Ram chip. The device offers all the necessary electronics for access from each side plus features which simplify system operation. There are special addresses at the top of the Dual Port Ram which, when written to, can be used to generate an interrupt for the 'other side'. Reading the corresponding address clears the interrupt. (See the manufactures device description for detailed information). The Dual Port Ram interface interrupt service routine reads the address to clear the interrupt and then sets an event for the 'MSGH' task.

CRM Semaphore Usage

```
pROBE+>qs

  Name        SMID         Count            TQ Len             Qtype
-------------------------------------------------------------------
  'RDA1' -#00030000     00000001         00000000             FIFO
  'WRA1' -#00040000     00000001         00000000             FIFO
  'RXC1' -#00050000     00000000         00000000             FIFO
  'TXC1' -#00060000     00000000         00000000             FIFO
  'init' -#00070000     00000001         00000000             Prio
```

There is only the one user semaphore 'init' which is basically used to prevent tasks writing to the Xilinx devices when they are not programmed (configured).

The pSOS system provided serial communications mechanisums on port 1 are controlled by semaphores transparrent to the user.

CRM Event usage

The messaging system interrupt sets an event flag on which the task 'msge' waits forever. Note that once triggered the 'msge' task will exhaust the message system queue then wait forever again. Should a series of messages be placed in the messaging system queue it is likely that the micro processor will exhaust the messages as part of a single activation. In this case the message event flag will be set when the micro goes to wait on the event. An immediate return will be made but the messaging system will be empty. This does not cause any problem to the system.

pSOS Message queue usage

There are no pSOS message queus used.

CRM files

ROOT.CC
: The first user task in the CRM.

PROG_XIL.CC
: This file contains the task which calls the modules to programme the Xilinx FPGA's.

MMUX_XIL.CC
: A place holder for the include file "MMUX_XIL.H" which contains the Xilinx configuration data for Mux case.

DMUX_XIL.CC
: A place holder for the include file "DMUX_XIL.H" which contains the Xilinx configuration data for Demux case.

CRM_INIT.CC
: Various initialisation functions.

MSGCRM.CC

This file contains the message handling task.

XBARHNDL.CC
The crossbar switch handling task.

SUCCISR.S
Dual port ram interrupt service routine.

XP_OUT.S
Diagnostic print.

FPGADATA.S
This file contains a place holder for the Sram store or Xilinx configuration data.

Include files
GLMACROS.H
GLTYPES.H
QMSG.H
These relate mainly to the dual port ram message system.

HARDWARE.H
This file satisfies compile requirements for the QMSG code. There is no relavent content.

SYS_CONF.H
pSOS configuration file.

DMUX_XIL.H
MMUX_XIL.H
These contain the Xilinx configurations.

CRM.H
Contains definitions for CRM hardware registers and some structures.

DCM Firmware Description

Overview

The functionality of the DCM module is described by reference to the SUCC messages it receives and the Tasks in the system. There may be message definitions in the code which are no longer used. These are not described pending their removal. In addition to the Tasks listed there are two interrupt service routines. One services the Dual Port Ram interface, the other all fpga interrupts.

Build information

Building the firmware is via the makefile located in the dcm_c directory. Refer to the BUILD document for full details of the tools and directories used.

DCM Tasks

The following is a pROBE query tasks, the out put shows a snap shot the after the DCM has been running, individual task descriptions follow.

pROBE+>qt

```
  Name      TID    Prio Mode Status    Susp?     Parameters              Ticks
-------------------------------------------------------------------------------
 'IDLE' -#00010000  00  2000 Running
 'ROOT' -#00020000  05  2000   Qwait          Q =  'DBGQ' -#000B0000   forever
 'FPGA' -#000C0000  FA  2005   Ready   YES
 'ROTC' -#000D0000  69  2005   Qwait          Q =  'ROTQ' -#00090000   forever
 'TOTC' -#000E0000  69  2005   Qwait          Q =  'TOTQ' -#000A0000   forever
 'MSGE' -#000F0000  64  2005  Evwait        EVENTS = 00000001         forever
 'TEST' -#00100000  32  0000 Created
```

'IDLE'

This task is a pSOS supplied task which soaks up idle time.

'ROOT'

After board initialisation and pSOS+ completes it's start up, this task is called as the first user task. It initialises pSOS+ devices for the pSOS+ clock and serial ports. It then performs the function of creating semaphores and queues. The user tasks as described below are created and started. Finally it lowers its priority to be the lowest user task and then waits for ever on a simple debug queue.

'FPGA'

Program Xilinx devices. This is run as part of the power up. There are two Xilinx sets of configuration data one for the devices on the DCM card and another set for the devices on the DMM cards. When first started the Xilinx configuration data is read from read only memory to sram. The devices are programmed and the task suspends. The task may be resumed and will then program the devices with data from the sram. This method permits the down load of different data to the sram prior to issuing reprogram Xilinx command.

'ROTC'

The 'ROTC' task basically services the 'ROTQ' queue. All Rot related actions are performed via the queue, this ensures that there is no asynchronous activity. The following messages are handled.

ROT_INTERRUPT

The assembly language interrupt service routine which handles all fpga generated interrupts places this message in the queue when required. The 'ROTC' task simply increments the DCM's view of ROT by the

required amount and if a newrot message buffer is available passes a newrot message up to the SUCC with the current ROT.
SetFPLed(3, RED); newrot message up to the SUCC

SYSTICK_INTERRUPT

This is loaded in the queue by the fpga interrupt service routine. If the UpdateRot flag is set then the DCM'S VIEW of ROT is updated. NB since the ROT_INTERRUPT message is queued before SYSTICK_INTERRUPT the old ROT will have been updated and passed up to the SUCC. Only on the next ROT_INTERRUPT will the new ROT be visible to the SUCC.
If the UpdateRot flag is not set then the ROT remaining registers are read and if non zero, then the equivalent value of remainder in sysclks is added to the current DCM view of ROT.
[It is by this mechanism we detect a discrepancy between the expected and actual number of STNCLKs between SYSTICK's].
If ChkRotMessageReceived is true the current DCM's View of ROT is in the message and returned to the SUCC.

eNewRot

The processing of this message is only to store away the SUCC/DCM message system message ready to be returned with NewRot information on the next ROT_INTERRUPT (TICK).
SetFPLed(3, GREEN); newrot message from the SUCC

eDcmSetRot

The 'ROTC' task simply stores the new value for ROT in holding registers and sets UpdateRot to TRUE. See SYSTICK_INTERRUPT actions for further processing.

eDcmCheckRot

The 'ROTC' task simply stores the message details ready to be processed on the next SYSTICK_INTERRUPT action.

eDcmConfiguration

The fpga RotIntRate registers are loaded with data from the configuration message. SysClksPerStnClk are stored to provide the multiplier required to convert the value of STN_CLK counts as used in the fpga hardware to EVN42 sysclks (In EVN42 all timing is based on sysclks from the beginning of the year, sysclks are at the rate of 32e+06 per second. The SUCC is blessed with good floating point math support, whereas the DCM is integer only. However to over come this problem and integer representation is used. A structure ITOT contains two 32 bit integers, integral seconds from the beginning of the year and the 'fractional' sysclks 0 -> 32000000.)

'TOTC'

The 'TOTC' task basically services the 'TOTQ' queue. All TOT related actions are performed via the queue, this ensures that there is no asynchronous activity. The following messages are handled.

SOT_INTERRUPT

There is a hardware signal which accompanies data which was recorded on an integral second of TOT. When received in the Delay memory modules the hardware address for this data is latched for each channel. The SOT_INTERRUPT is used to inform the micro processor that new addresses are available. If a previous eDcmSotTot message has been received the addresses are stored. (From this point on the location of any data is deterministic). Only the single snapshot address is required for data location purposes. However for hardware debug, a series of snap shots have been captured and stored. The code for this remains active.

eDcmSotTot

When this message is processed Update Tot is set TRUE and the new Tot information is stored ready to be used in processing the SOT_INTERRUPT as above.

BOCF_INTERRUPT

This is handled by calling BOCFInterruptHandler(). The actual processing is dependent on the state of scan. The cases below are covered. eBocf_S_TEST :

eBocf_S_INSCAN :

In these the BOCF header Hardware holding FIFO's are loaded with data for the frame after next. The Hardware holding registers for the astronomic data addresses and the delay accumulator settings are loaded for the next frame. A request is made for more BOCF header data from the SUCC.

eBocf_S_RUNDOWN :

See eBocf_S_IDLE.

eBocf_S_IDLE :

In eBocf_S_RUNDOWN and eBocf_S_IDLE the same actions as above are performed with a default set of delay data. No request is made for further BOCF data from the SUCC.

'MSGE'

This task actions the messages received from the SUCC as described above.

Interrupt Service Routines

Dual Port Ram interface interrupt service routine

The communication between the SUCC and the DCM is via a Dual Port Ram physically implemented in the IDT7025 Dual Port Ram chip. The device offers all the necessary electronics for access from each side plus features which simplify system operation. There are special addresses at the top of the Dual Port Ram which, when written to, can be used to generate an interrupt for the 'other side'. Reading their corresponding address clears the interrupt. (See the manufactures device description for detailed information). The Dual Port Ram interface interrupt service routine reads the address to clear the interrupt and then places a message on the 'MSGQ' system message queue.

Fpga interrupt service routine

All fpga interrupt sources set bits in one status register and only one interrupt line into the processor is used. When activated the service routine reads the status register and issues queue messages as required. The order of processing is as below.

```
Status Register bit     Message Queued
ROT_BIT                 ROT_INTERRUPT
BOCF_BIT                BOCF_INTERRUPT
SOT_BIT                 SOT_INTERRUPT
SYSTICK_BIT             SYSTICK_INTERRUPT
ROLLOVER_BIT            ROLLOVER_INTERRUPT ( Not Used )
```

DCM / SUCC message types.

The following message types are implemented in the DCM firmware. A brief explanation for the processing of each message follows. All messages from the SUCC are initially handled by the task 'MSGE'. Some require additional processing by specialised tasks. In these cases the message details are forwarded to the relevant task.

Special note for Delay Memory access.
The original concept for the Delay Module expected micro processor access to the Delay memory to be available. Xilinx design development necessitated removal of this facility. There are some references in EVN42 which imply access, these should not be used.

eDcmLoadRam:

This is restricted to Micro Sram access.

eDcmReadRam:

This is restricted to Micro Sram access.

eProgFpga:

Normal system operation for programming the Xilinx devices from power up by starting the 'FPGA' task which copies the default Xilinx configuration information from EPROM into holding areas in Sram. It then programs/configures the Xilinx devices using data from the holding areas. Separate data areas hold 'dcm_fpga_data' and 'dmm_fpga_data'. When complete the task suspends. If it is resumed it skips the copy from EPROM and programs/configures the Xiinx devices using data from the holding areas.
The eProgFpga message provides the facility to enable the C3 to reprogramme the Xilinx FPGA's using a configuration file down loaded into Sram. This has been used in early development but is not a routine operation.

eDcmConfiguration:

The control registers in the Xilinx devices are loaded with data derived from the configuration message. In general the data will be loaded as is, however counters are loaded with the value minus one. (i.e. HeaderLength, HeaderMultiplier and RotIntRate.) A copy is made of the new configuration and some of the configuration is performed directly in the message handling task and the rest is done in the 'ROTC' task. A message is sent on the 'ROTQ' queue containing eDcmConfiguration as the message type and a pointer to the local configuration.

eDcmSetRot:

This is passed to the 'ROTC' task via the 'ROTQ' queue. 'ROTC' stores the new rot information and sets an internal flag which indicates that the new ROT should be set on the next SYSTICK. (Until the next SYSTICK the DCM will continue with the current time base for each ROTTICK). The message is returned to the SUCC immediately.

eDcmCheckRot:

This is passed to the 'ROTC' task via the 'ROTQ' queue. 'ROTC' stores the message details and sets an internal flag which indicates that on the next SYSTICK 'ROTC' should return its current idea of ROT. The message is returned on the next SYSTICK to the SUCC with the current Rot appended to the message. (The message also contains the expected ROT but this is not referenced by the DCM). Only one CheckRot message should be out standing at any time. If more are sent before completion the temporary storage for message details will be over written resulting in message buffer loss

eDcmTxStartRot:

The message is handled by a call to 'BOCFInitScan' found in bocf.cc and is returned to the SUCC immediately. Refer to 'BOCFInitScan' for details.

pSuccCmd->MsgBody.DcmTxStartRot.StartRot,
pSuccCmd->MsgBody.DcmTxStartRot.RotIncrement,
pSuccCmd->MsgBody.DcmTxStartRot.EndRot,
pSuccCmd->MsgBody.DcmTxStartRot.Systicks2Start,
pSuccCmd->MsgBody.DcmTxStartRot.BOCFOffsetReg);

eNewRot:

This is passed to the 'ROTC' task via the 'ROTQ' queue. 'ROTC' stores the message details and sets an internal flag which indicates that a new empty ROT message has been received. 'ROTC' should then return its idea of ROT on the next ROTTICK. (The DCM maintains its idea of ROT independent of this eNewRot packet passing mechanism. Should the SUCC not finish its processing of the eNewRot packet and pass the empty packet back before the next ROTTICK there will be no impact on the ROT timing.)

eDcmSotTot:

This is passed to the 'TOTC' task via the 'TOTQ' queue. 'TOTC' stores the new SOT information and sets an internal flag which indicates that the new information should be used at the next SOT. Each DMM RX_In_Ctrl register is loaded with RX_DISABLE then RX_ENABLE which resets the hardware ready for a new scan. Note that during this period there should be no transmit activity from the CRM to the DCM.

eBocfData:

See eBocfReq.

eBocfReq:

Both eBocfReq and eBocfReq are handled by calls to 'BOCFDataHandler'. Refer to 'BOCFDataHandler' for details.

eFreeze

On receipt this disables the receive and transmit mechanisms. There is little value in this now that access to the delay memory access is not available to the micro processor.

eAbort

This is used to terminate a scan before the originally programmed time. The BOCF handling state is set to run down. It is essential to use this to ensure that the BOCF handling is in the correct state for subsequent scans.

eDebug

A simple early diagnostic which returns the Debug message packet with data set to Ping.

eFlushDelayRam

This is not available with the current DMM Xilinx.

eReadTotData:

Not available.

DCM Queues

```
pROBE+>qq

  Name       QID      TQ Len    MQ Len   MQ Limit  Buf Src  Qtype Variable
-------------------------------------------------------------------------
  'ROTQ' -#00090000  00000001  00000000  00000014   Private  FIFO   No
  'TOTQ' -#000A0000  00000001  00000000  00000014   Private  FIFO   No
  'DBGQ' -#000B0000  00000001  00000000  00000014   Private  FIFO   No

Sys-pool total = 00000064
Sys-pool free  = 00000028
```

The firmware uses a set of pSOS fixed length queues to provide standard communication facilities.

'ROTQ'

Read by task 'ROTC', and written to by the fpga interrupt service routine and the SUCC message handling task 'MSGE'.

'TOTQ'

Read by task 'TOTC', and written to by the fpga interrupt service routine and the SUCC message handling task 'MSGE'.

'DBGQ'

Read by task 'ROOT', and was used during debug but is no longer.

DCM Semaphores

pROBE+>qs

```
  Name       SMID        Count           TQ Len            Qtype
-------------------------------------------------------------------
  'RDA1' -#00030000     00000001        00000000             FIFO
  'WRA1' -#00040000     00000001        00000000             FIFO
  'RXC1' -#00050000     00000000        00000000             FIFO
  'TXC1' -#00060000     00000000        00000000             FIFO
  'INIT' -#00070000     00000001        00000000             FIFO
  'MSGQ' -#00080000     00000001        00000000             FIFO
```

pSOS+ semaphores 'RDA1' 'WRA1' 'RXC1' 'TXC1' are created and maintained transparently by the pSOSSystem serial port drivers.

'INIT'

Combined with the task priority structure this semaphore provides protection around the programming of Xilinx devices. It prevents tasks writing to an unprogrammed device and suffering a bus error.

'MSGQ'

This provides security for tasks performing actions on the messaging system. It is transparent to normal user actions.

pROBE+>qo

```
Resident Objects:      Name      ID         Type       Access
-------------------------------------------------------------
                      'RN#0' -#00000000     REGION      Local
                      'IDLE' -#00010000     TASK        Local
                      'ROOT' -#00020000     TASK        Local
                      'RDA1' -#00030000     SEMAPHORE   Local
                      'WRA1' -#00040000     SEMAPHORE   Local
                      'RXC1' -#00050000     SEMAPHORE   Local
                      'TXC1' -#00060000     SEMAPHORE   Local
                      'INIT' -#00070000     SEMAPHORE   Local
                      'MSGQ' -#00080000     SEMAPHORE   Local
                      'ROTQ' -#00090000     QUEUE       Local
                      'TOTQ' -#000A0000     QUEUE       Local
                      'DBGQ' -#000B0000     QUEUE       Local
                      'FPGA' -#000C0000     TASK        Local
                      'ROTC' -#000D0000     TASK        Local
                      'TOTC' -#000E0000     TASK        Local
                      'MSGE' -#000F0000     TASK        Local
                      'TEST' -#00100000     TASK        Local
```

DCM files

ROOT.CC
: The first user task in the DCM.

PROGFPGA.CC
: This file contains the task which calls the modules to programme the FPGA's.

PDMMFPGA.CC
: This module programmes the Delay Memory Module FPGAs.

PDCMFPGA.CC
: This module programmes the Delay Controller FPGA.

FPGAINC.CC
: This file just contains place holders for "dcm_fpga.h" and "dmm_fpga.h".

ROTC.CC
Task ROTC is contained in this file.

TOTC.CC
Task TOTC is contained in this file.

MSGHNDL.CC
The message system handling task.

TEST.CC
A development test routine which is not used in operational firmware.

FPGADATA.S
The temporary sram storage for FPGA data is defined in this file.

DCMISR.S
The FPGA interrupt service code.

SUCCISR.S
The dual port ram interrupt service code.

BOCF.CC
This file contains the functions which control the BOCF related actions.

QMSG.CC
The dual port ram message system code.

Include files
GLMACROS.H
GLTYPES.H
QMSG.H
These relate mainly to the dual port ram message system.

HARDWARE.H
This file satisfies compile requirements for the QMSG code. There is no relavent content.

DCM.H
Contains definitions for DCM hardware registers and some structures.

SYS_CONF.H
pSOS configuration file.

CFHEADER.H
This defines the correlator frame header content.
DCM_FPGA.H
DMM_FPGA.H
These contain the Xilinx configuration files which are used to programme the Xilinx devices.

DCMFP.H
This header file contains some (not all) function prototypes.

DCM Front Panel LED's

The four tricolour front panel LED's are used to provide some information about the state of the DCM.

The LED's are programmable and numbered 2 -> 5. On the DCM prototype front panel the top LED is an indictor that the DCM board Xilinx devices are programmed. LED2 is the second LED down from the top and is the first software controlled LED.

After power up BOCFInitHandler() is called and sets all LED's to OFF.

LED2
This is set GREEN each time the BOCF header data used is thought to be valid. If the BOCF header pipeline appears to be broken and default header data is used inside a scan then LED2 is set RED.

LED3
When the DCM sends a new ROT message up to the SUCC LED3 is set RED.
When the SUCC responds with an empty new ROT message packet LED3 is set GREEN.
Under normal system operation one should see a very faint red flicker if ROT is running.

LED4
BOCFInitScan() sets this LED YELLOW when it actions the start of scan message. The purpose is to provide a signal for an operator to prime a logic analyser.
On receipt of BOCF Data back from SUCC in scan, LED4 is set GREEN. It indicates that the first BOCF interrupt has occurred and the SUCC has provided new set of headers. This may also be used as a signal for an operator to prime a logic analyser.
At end of scan LED4 is set RED.

LED5
On sending requests for BOCF Data to the SUCC LED5 is set RED.
On receipt of BOCF Data back from SUCC LED5 is set GREEN.
Under normal system operation during a scan one should see a very faint red flicker at the rate of BOCF.

Workarounds

BOCF SYSTICK ALIGNMENT

Using the BOCF offset to compensate for the 11 STNCLKS pipeline delays in the DCM/DMM.

The C3 should send a BOCF offset which will delay the first Header by 1 second minus 11STNCLKS in the units of 32MHz SYSCLKS. In the case of half time operation SYSTICKS happen on every half ROT second. So the offset needs to be configured possibly to 0.5 * the number of clocks.

The "scan data to scan interval" which is used in the test system to give the Orange Led period prior to the first BOCF, must not be shorter than 2 seconds. In the test system we use 3 seconds (in sysclks). This time allows the DCM SW to account for the extra second necessary befor the first BOCF.

The "scan data to scan interval" must be extended in ROT time if running double or quad time to allow for a SYSTICK to start the BOCF offset down counter such that on the following SYSTICK the scan can start. Note that SYSTICKS are of the order of one per real time second.

The BOCF offset must also be extended to allow for the count of STNCLKS from the first SYSTICK to the Start of scan SYSTICK. N.B. The BOCF offset counter is a 32 bit counter and runs at STNCLK rate. The BOCF Offset as in the SUMISC parameters message is in sysclks time.

The ROT TOT offset value in SUMISC parameters should have the value of half the Deskew ram in sysclks (128/2 frames) plus half the delay buffer (4 * 1024 * 1024)/2 * the value in sysclks of a STNCLK.

SUIM / DCM setup.

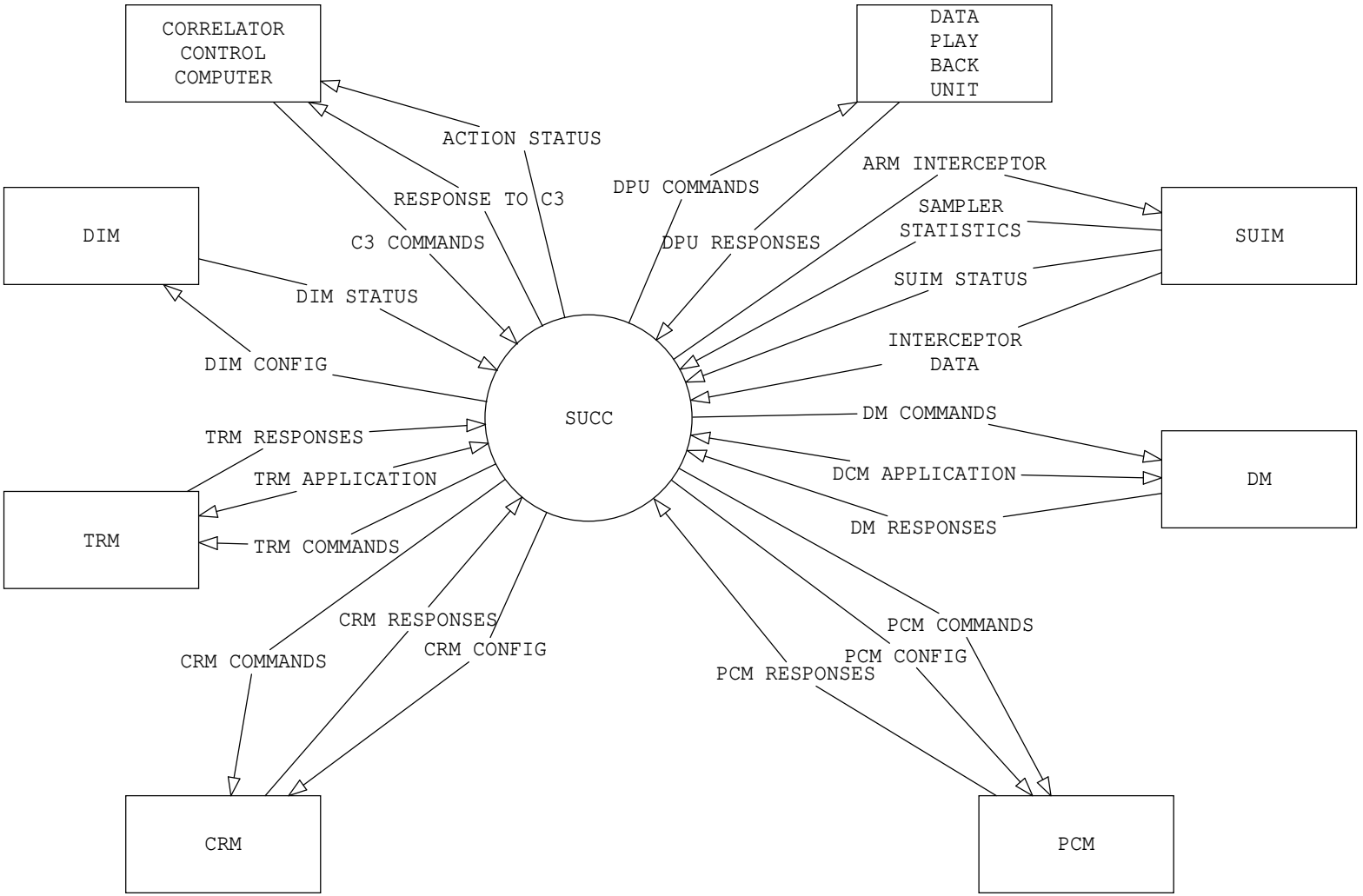
A problem can arise when changing from a slow STNCLK to a fast STNCLK. e.g.
Config A is 2 MHz STNCLK with DCM set for 50 BOCF's per realtime second and Config B is 32 MHz STNCLK with any BOCF and ROT Tick rate. The ROT tick rate will be of the order of 16 or 32 per realtime second for each case.
We assume that configuration A is running with BOCF and ROT intrrupts then we wish to change to configuration B.
The original method of changing would be to set the SUIM to 32 MHz (via its intermediate state as required by SUIM hardware). Then set the DCM configuration for Configuration B.
This has the problem that the DCM sees the a change in STNCLK from 2 to 32 MHz which increases both the ROT tick and BOCF interrupt rates by a factor of 16. The result is a BOCF interrupt rate of 800 and ROT Ticks well in to hundreds as well. While the DCM is trying to cope with all of this extra activity it does not get around to reconfiguring. So it just appears to hang.
To prevent this happening reconfigure the DCM for the higher rates before changing the SUIM setting.

Yourdon Design

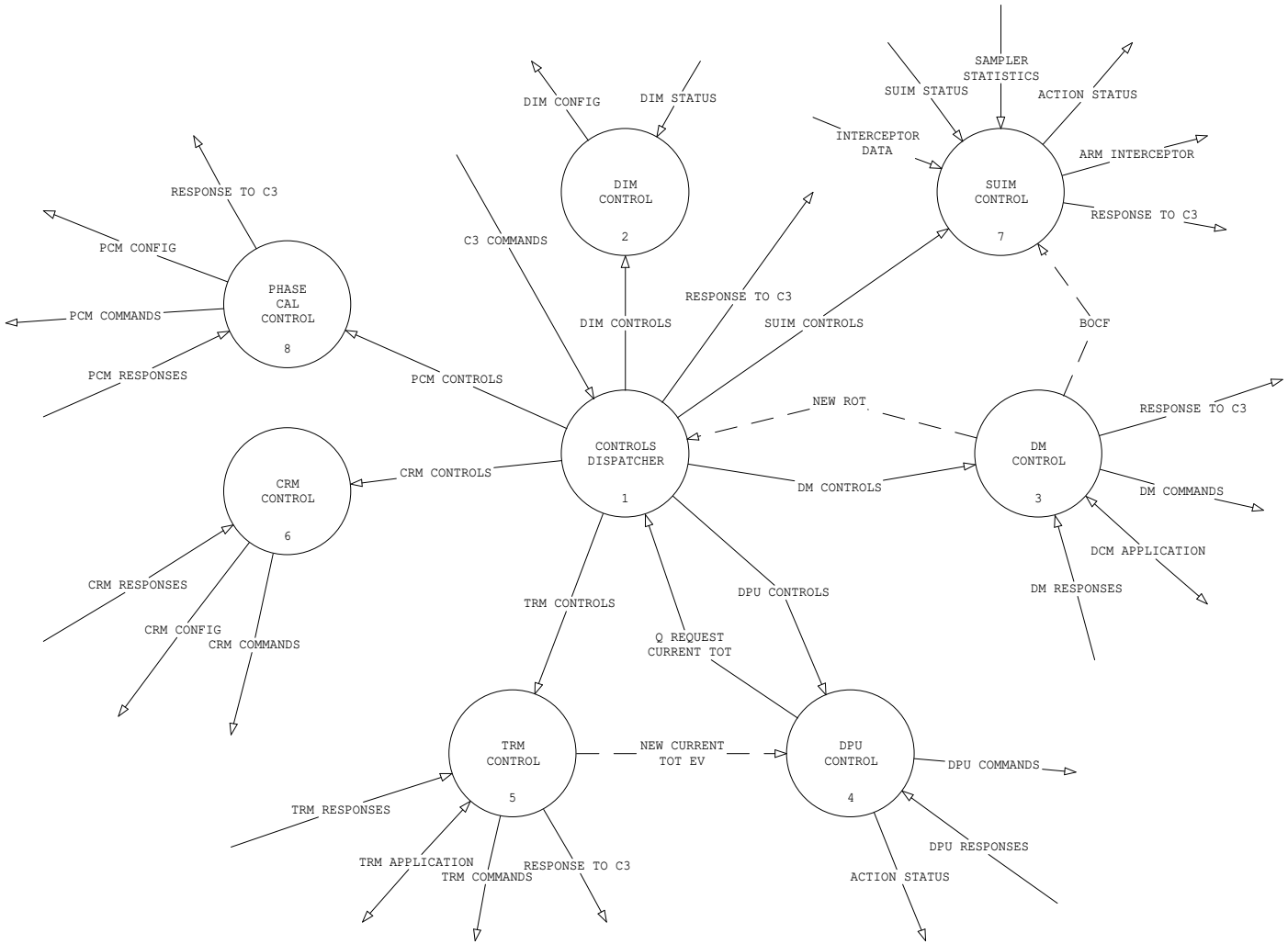
The Yourdon diagrams show the intended structure for each major unit. Due to changes in the available functions, some details are now a little out of date. . In some cases the pspecs are not complete. However as an over view the Yourdon diagrams still provide a good insight into the operation of the firmware.

SUCC Yourdon Diagrams:

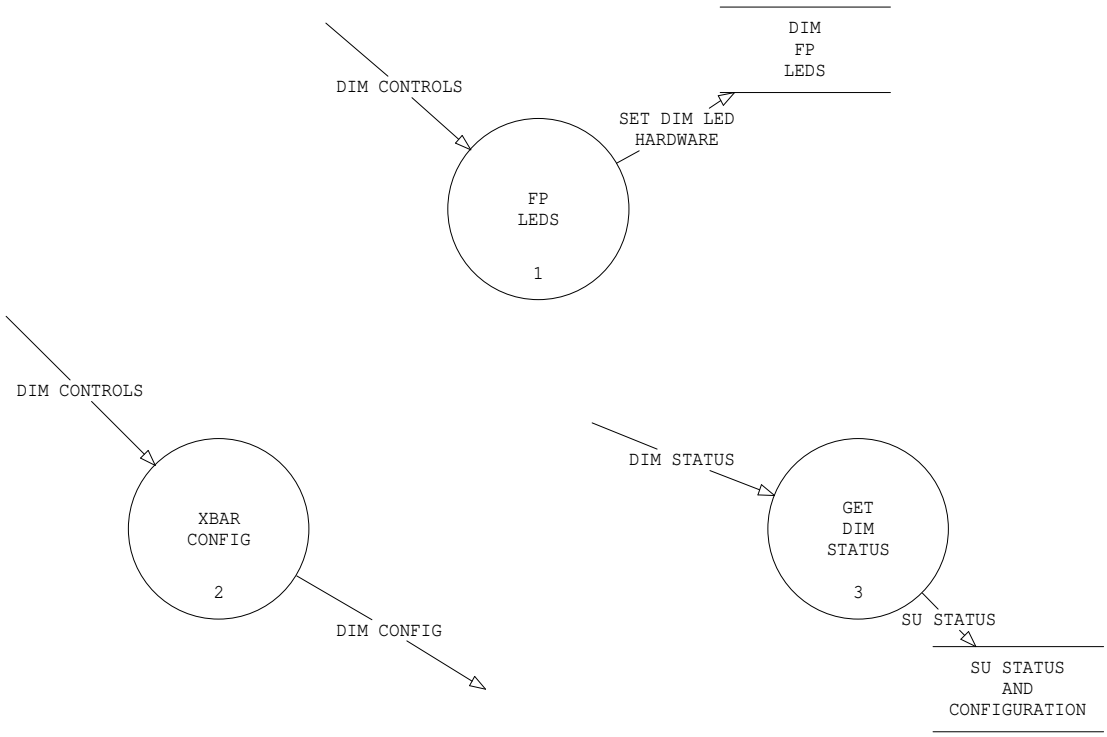
Context diagram for the SUCC.



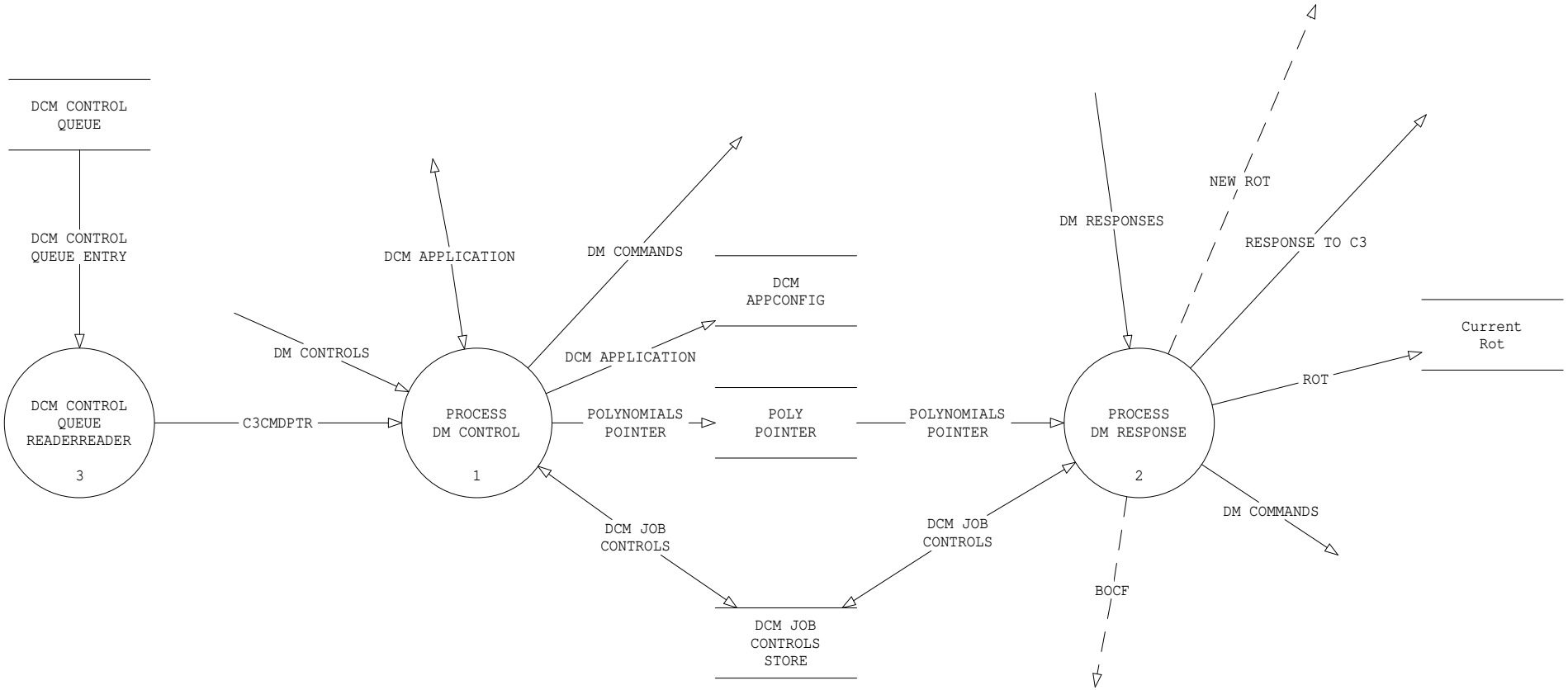
SUCC.


DIM CONFIG
DIM STATUS
DIM CONTROL
2
SAMPLER STATISTICS
SUIM STATUS
ACTION STATUS
INTERCEPTOR DATA
ARM INTERCEPTOR
SUIM CONTROL
7
RESPONSE TO C3
RESPONSE TO C3
PCM CONFIG
PHASE CAL CONTROL
8
PCM COMMANDS
PCM RESPONSES
C3 COMMANDS
DIM CONTROLS
RESPONSE TO C3
SUIM CONTROLS
BOCF
PCM CONTROLS
NEW ROT
CONTROLS DISPATCHER
1
RESPONSE TO C3
DM CONTROL
3
DM COMMANDS
CRM CONTROLS
DM CONTROLS
CRM CONTROL
6
DCM APPLICATION
DM RESPONSES
CRM RESPONSES
CRM CONFIG
CRM COMMANDS
TRM CONTROLS
DPU CONTROLS
Q REQUEST CURRENT TOT
TRM CONTROL
5
NEW CURRENT TOT EV
DPU CONTROL
4
DPU COMMANDS
TRM RESPONSES
DPU RESPONSES
TRM APPLICATION
TRM COMMANDS
RESPONSE TO C3
ACTION STATUS

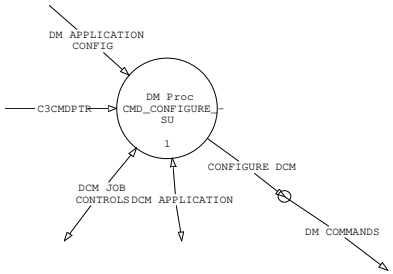
DIM CONTROL.

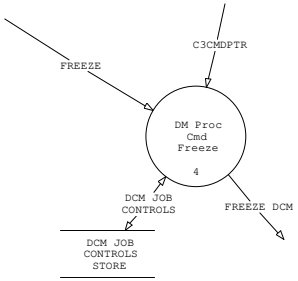

DIM
FP
LEDS
DIM CONTROLS
SET DIM LED
HARDWARE
FP
LEDS
1
DIM CONTROLS
XBAR
CONFIG
2
DIM CONFIG
DIM STATUS
GET
DIM
STATUS
3
SU STATUS
SU STATUS
AND
CONFIGURATION

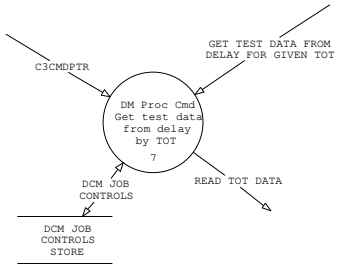
DM CONTROL.

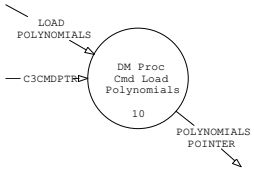

DCM CONTROL QUEUE
DCM CONTROL QUEUE ENTRY
DCM CONTROL QUEUE READERREADER
3
C3CMDPTR
DM CONTROLS
DCM APPLICATION
PROCESS DM CONTROL
1
DM COMMANDS
DCM APPCONFIG
DCM APPLICATION
POLYNOMIALS POINTER
POLY POINTER
POLYNOMIALS POINTER
DCM JOB CONTROLS
DCM JOB CONTROLS STORE
DCM JOB CONTROLS
DM RESPONSES
PROCESS DM RESPONSE
2
NEW ROT
RESPONSE TO C3
Current Rot
ROT
DM COMMANDS
BOCF

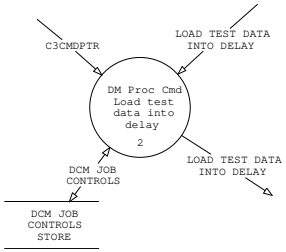
PROCESS DM CONTROL.

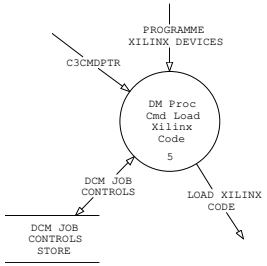


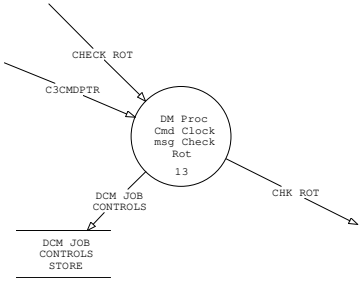


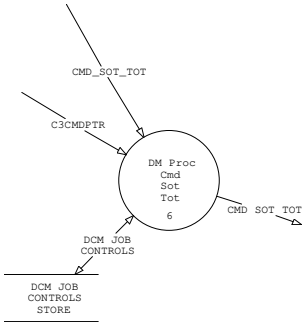


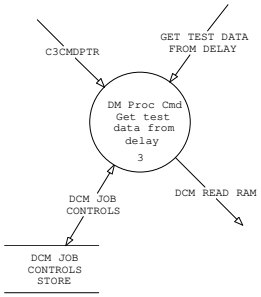


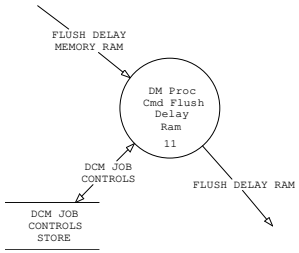


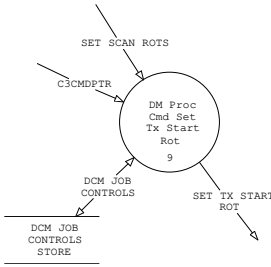


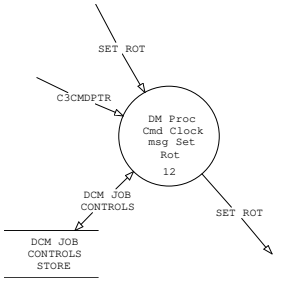




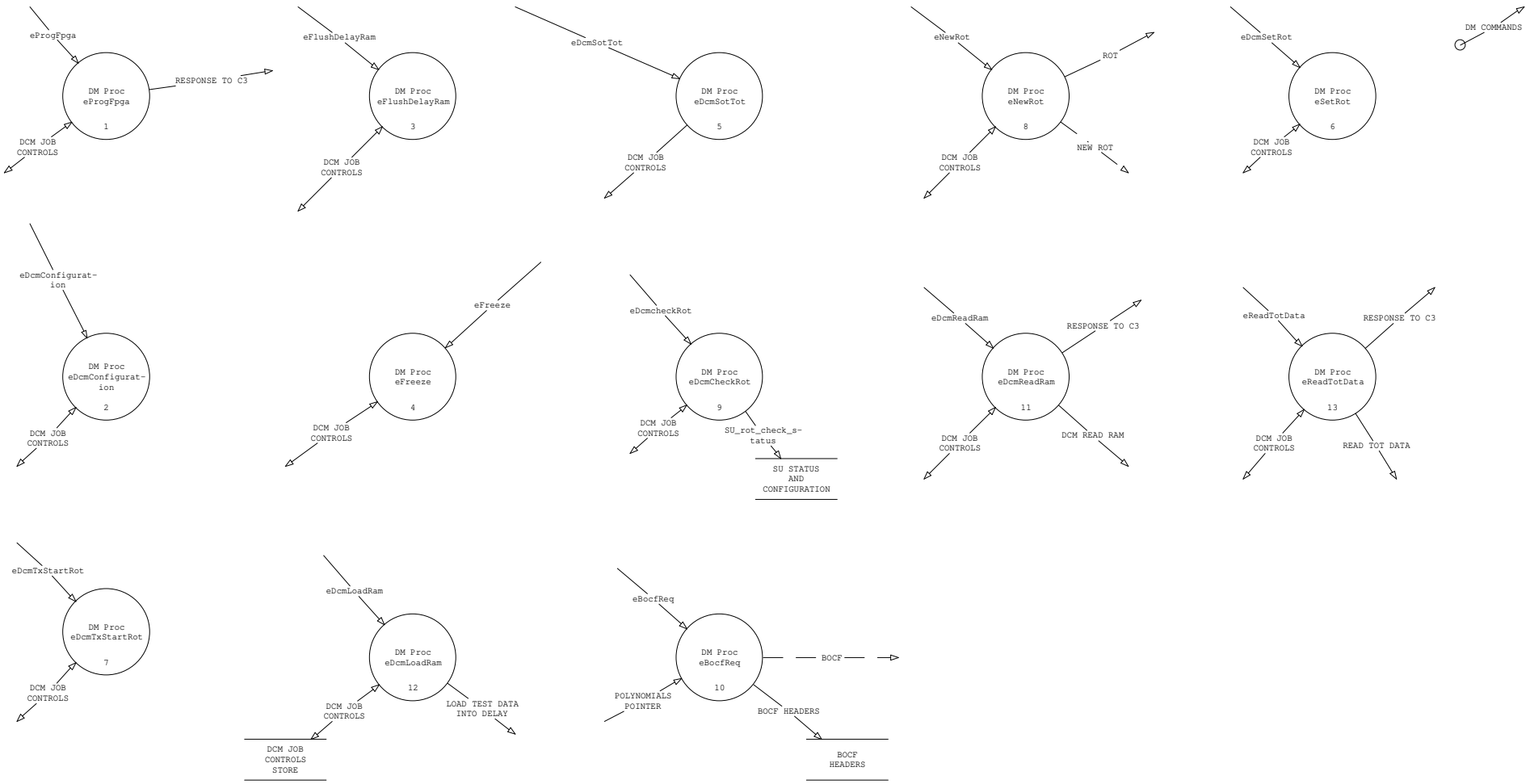




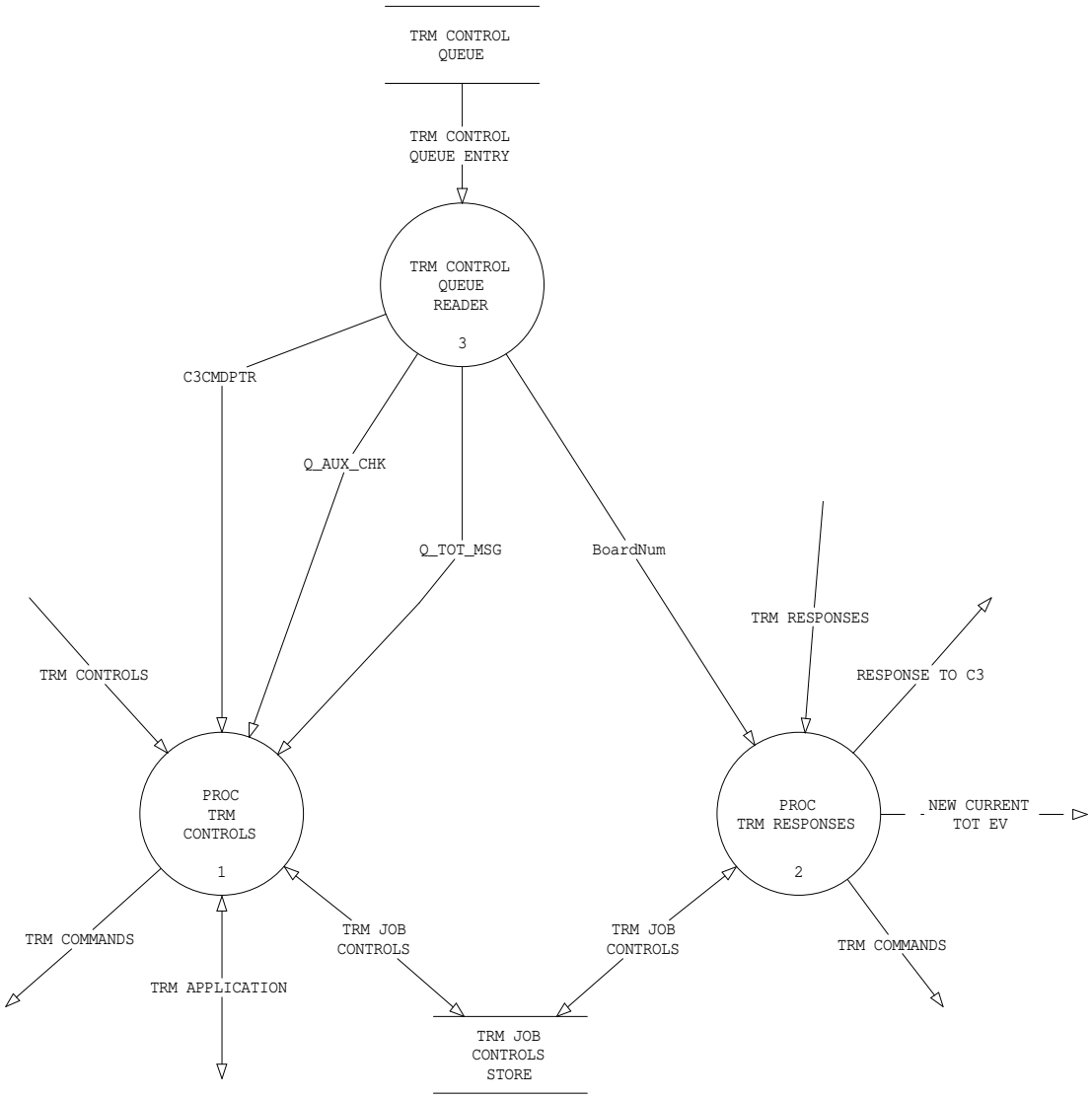




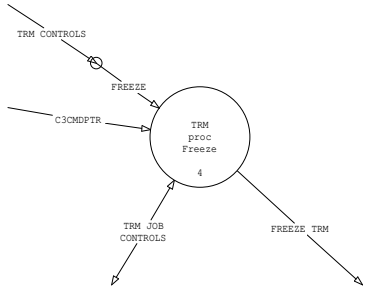
PROCESS DM RESPONSE.

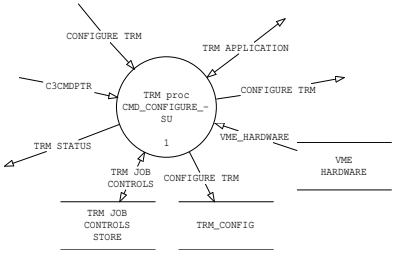


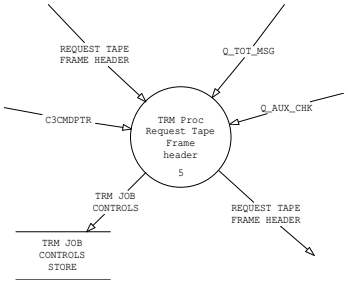
TRM CONTROL.

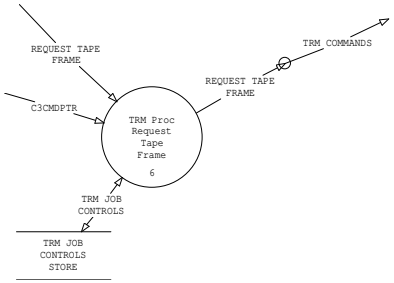

TRM CONTROL
QUEUE
TRM CONTROL
QUEUE ENTRY
TRM CONTROL
QUEUE
READER
3
C3CMDPTR
Q_AUX_CHK
Q_TOT_MSG
BoardNum
TRM RESPONSES
TRM CONTROLS
RESPONSE TO C3
PROC
TRM
CONTROLS
1
PROC
TRM RESPONSES
2
NEW CURRENT
TOT EV
TRM COMMANDS
TRM APPLICATION
TRM JOB
CONTROLS
TRM JOB
CONTROLS
TRM COMMANDS
TRM JOB
CONTROLS
STORE

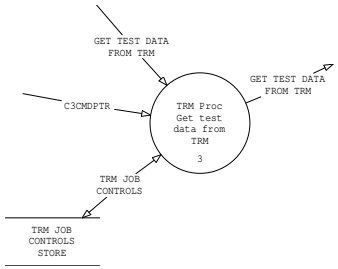
PROC TRM CONTROLS.

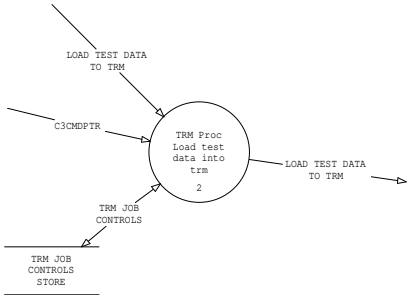


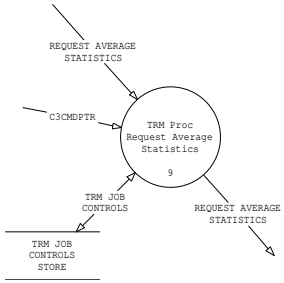


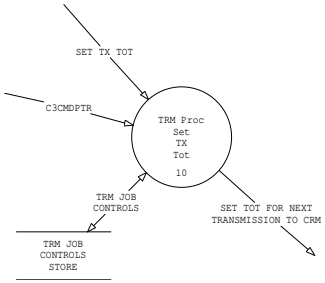


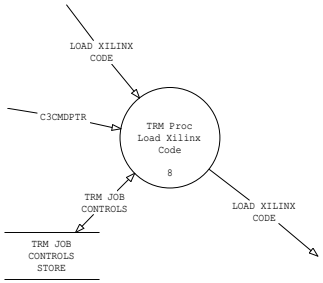


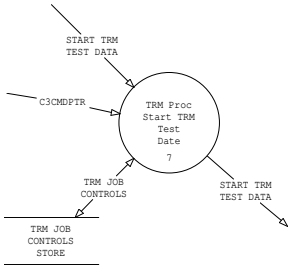


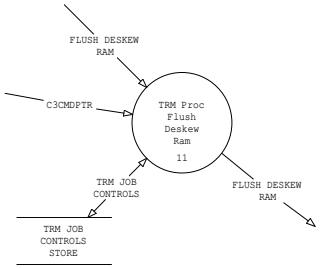






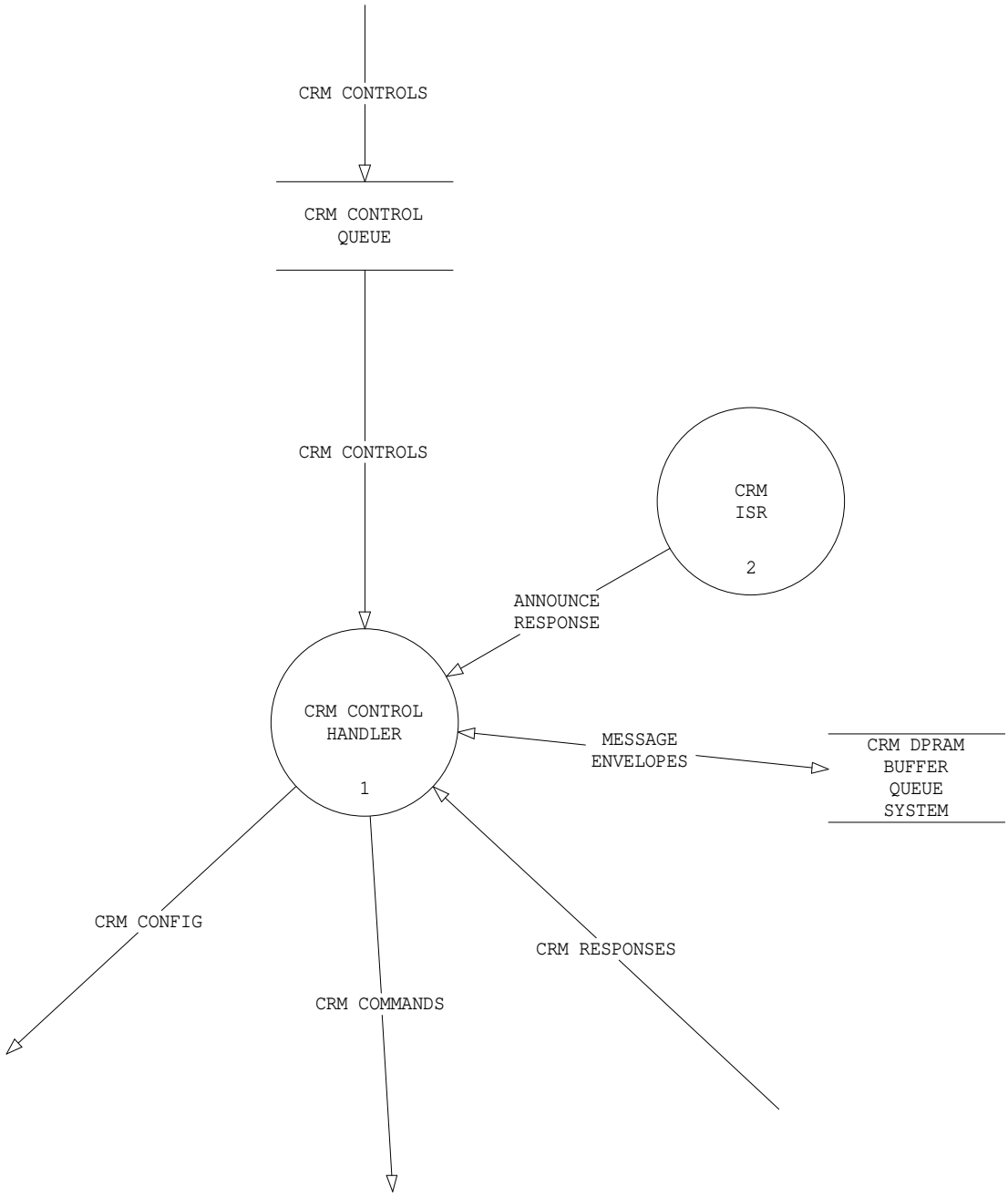




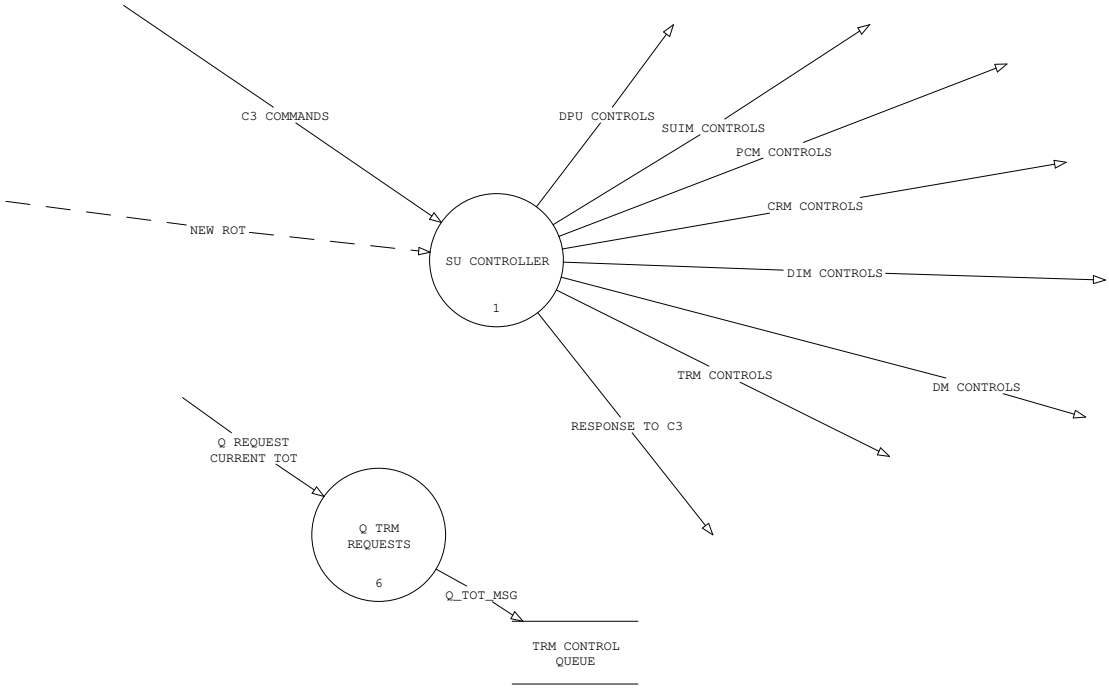


PROC TRM RESPONSES.

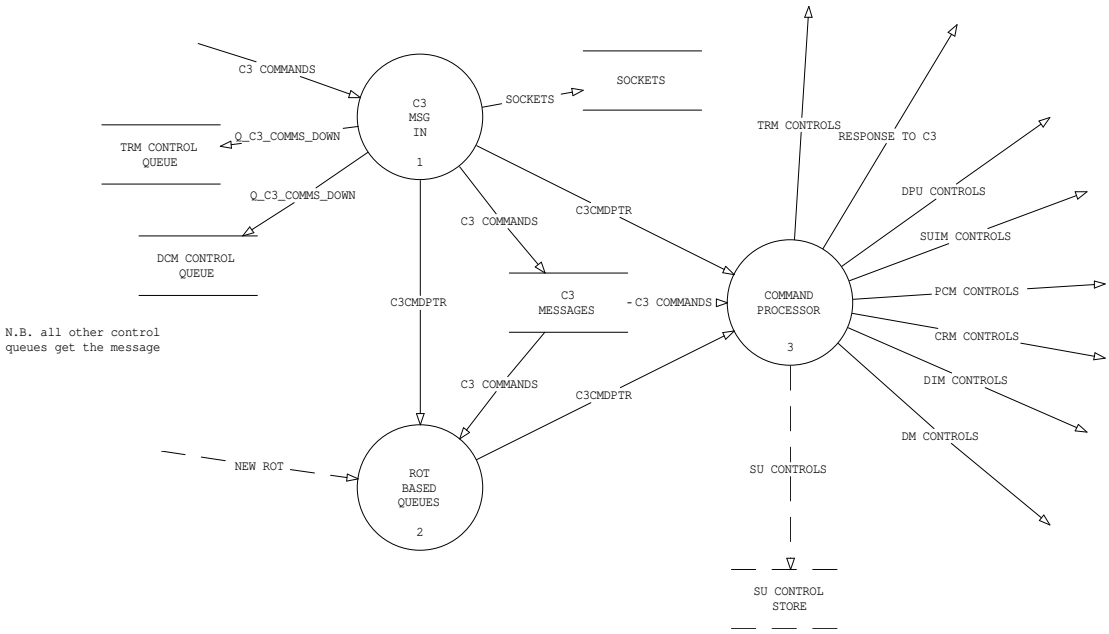
CRM CONTROL.


CRM CONTROLS
CRM CONTROL QUEUE
CRM CONTROLS
CRM ISR
2
ANNOUNCE RESPONSE
CRM CONTROL HANDLER
1
MESSAGE ENVELOPES
CRM DPRAM BUFFER QUEUE SYSTEM
CRM CONFIG
CRM COMMANDS
CRM RESPONSES

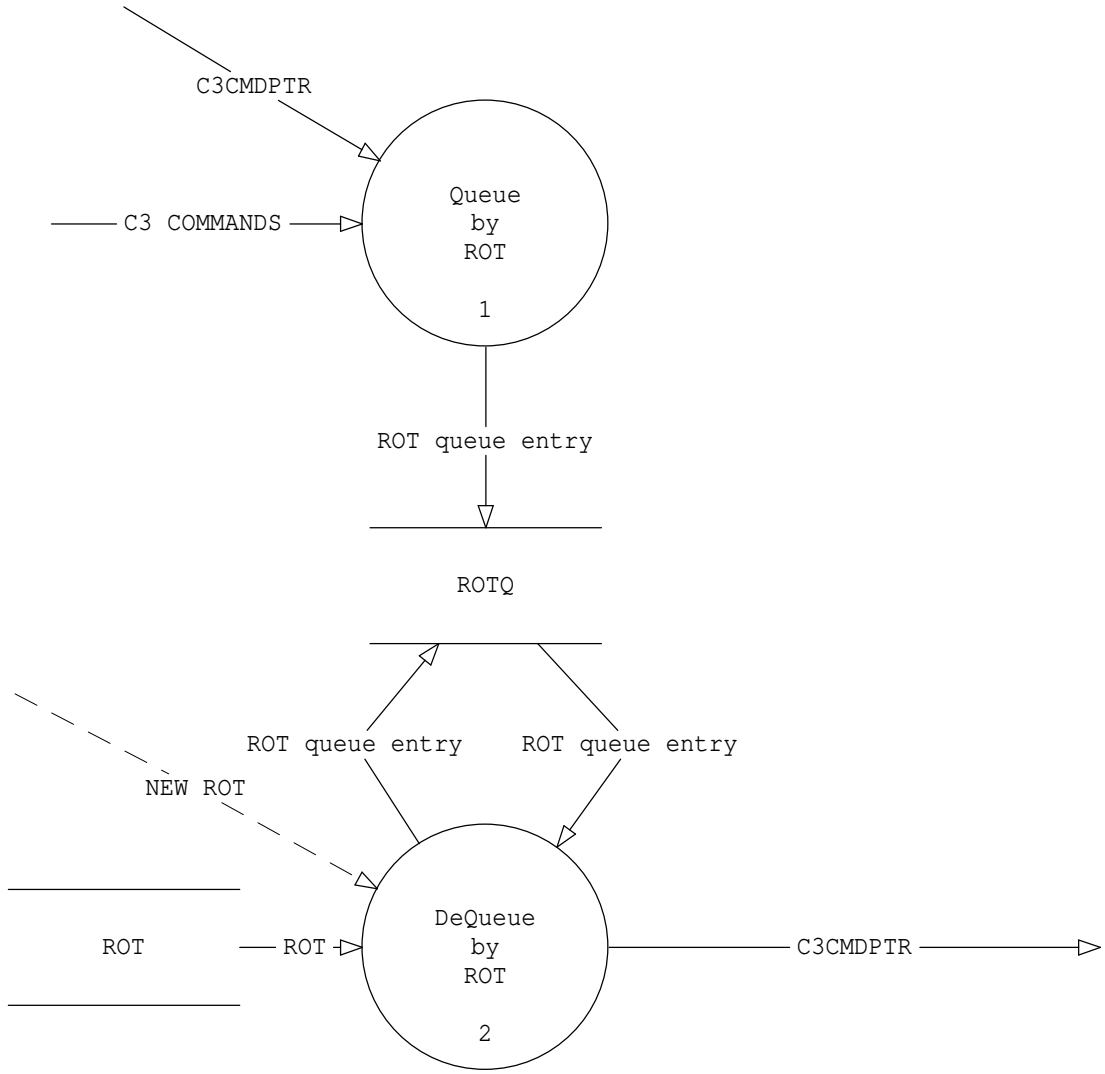
CONTROLS DISPATCHER.



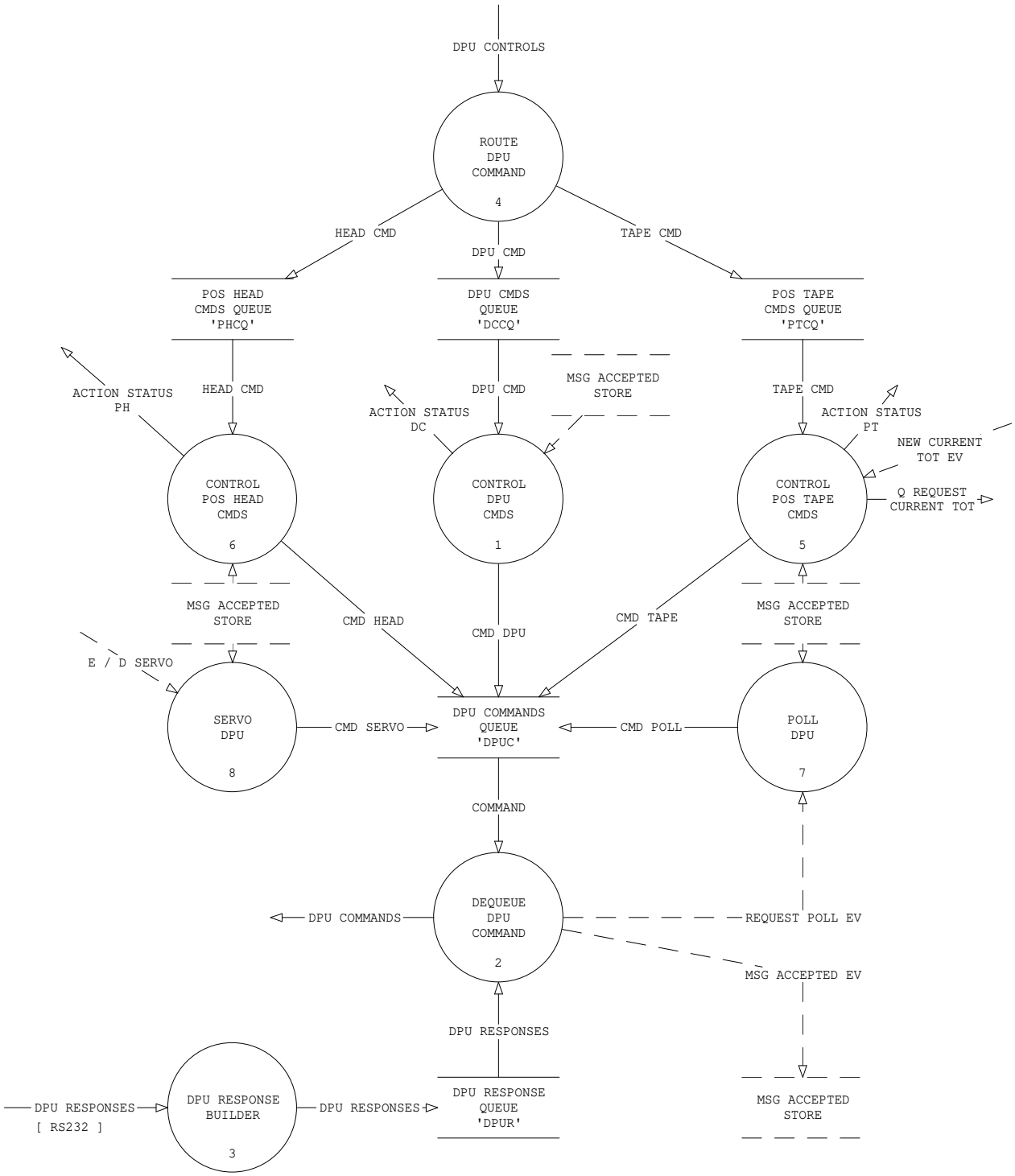
SU CONTROLLER.



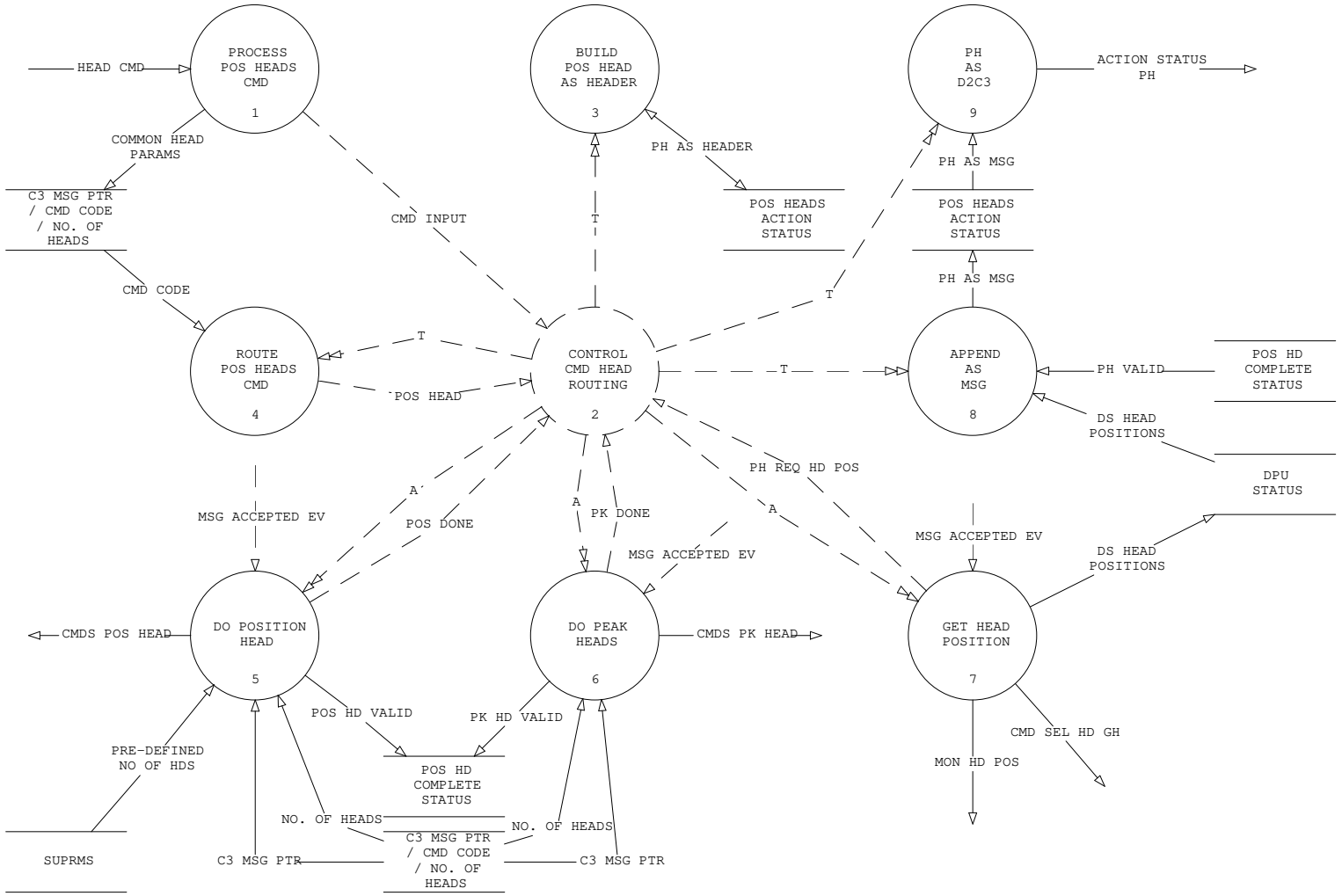
ROT BASED QUEUES.



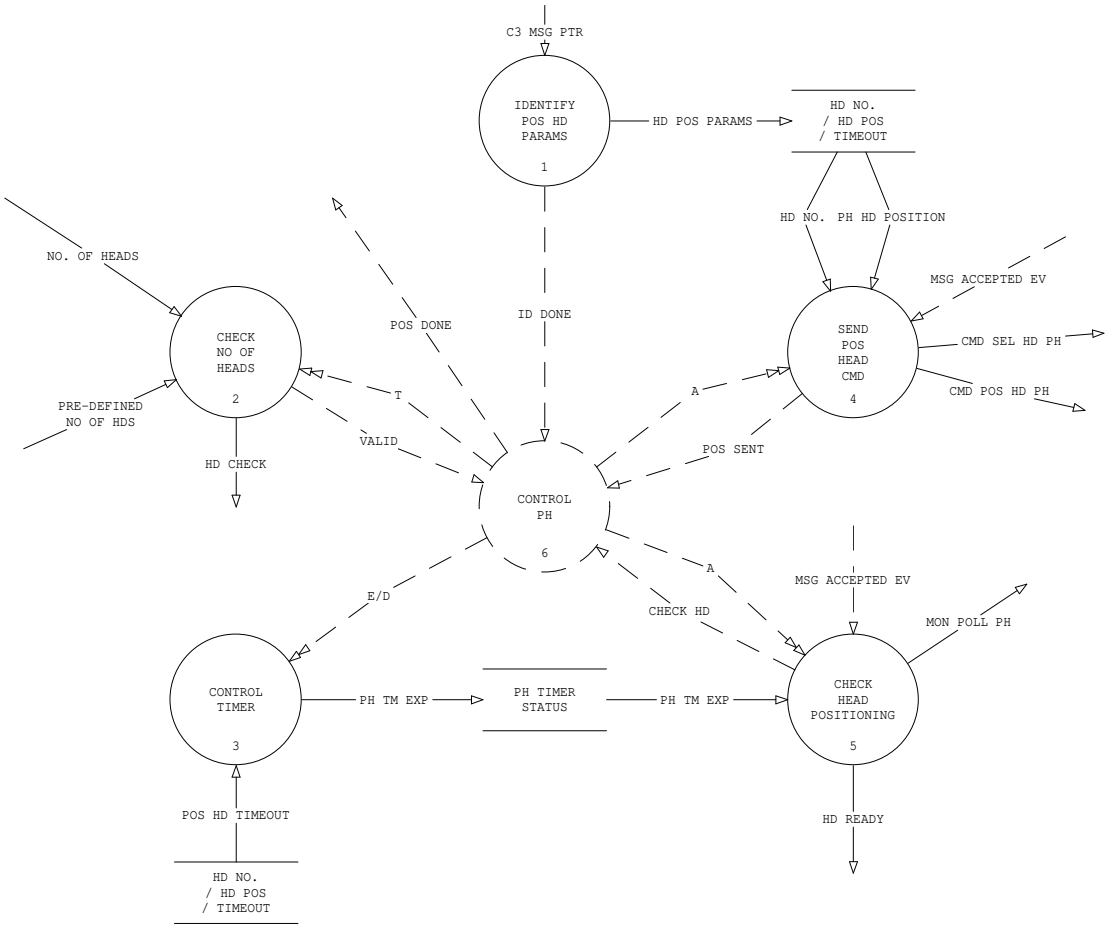
DPU CONTROL.


DPU CONTROLS
ROUTE DPU COMMAND 4
HEAD CMD
DPU CMD
TAPE CMD
POS HEAD CMDS QUEUE 'PHCQ'
DPU CMDS QUEUE 'DCCQ'
POS TAPE CMDS QUEUE 'PTCQ'
ACTION STATUS PH
HEAD CMD
ACTION STATUS DC
DPU CMD
MSG ACCEPTED STORE
TAPE CMD
ACTION STATUS PT
NEW CURRENT TOT EV
Q REQUEST CURRENT TOT
CONTROL POS HEAD CMDS 6
CONTROL DPU CMDS 1
CONTROL POS TAPE CMDS 5
MSG ACCEPTED STORE
CMD HEAD
CMD DPU
CMD TAPE
MSG ACCEPTED STORE
E / D SERVO
SERVO DPU 8
CMD SERVO
DPU COMMANDS QUEUE 'DPUC'
CMD POLL
POLL DPU 7
COMMAND
DEQUEUE DPU COMMAND 2
DPU COMMANDS
REQUEST POLL EV
MSG ACCEPTED EV
DPU RESPONSES
DPU RESPONSES [RS232]
DPU RESPONSE BUILDER 3
DPU RESPONSES
DPU RESPONSE QUEUE 'DPUR'
MSG ACCEPTED STORE

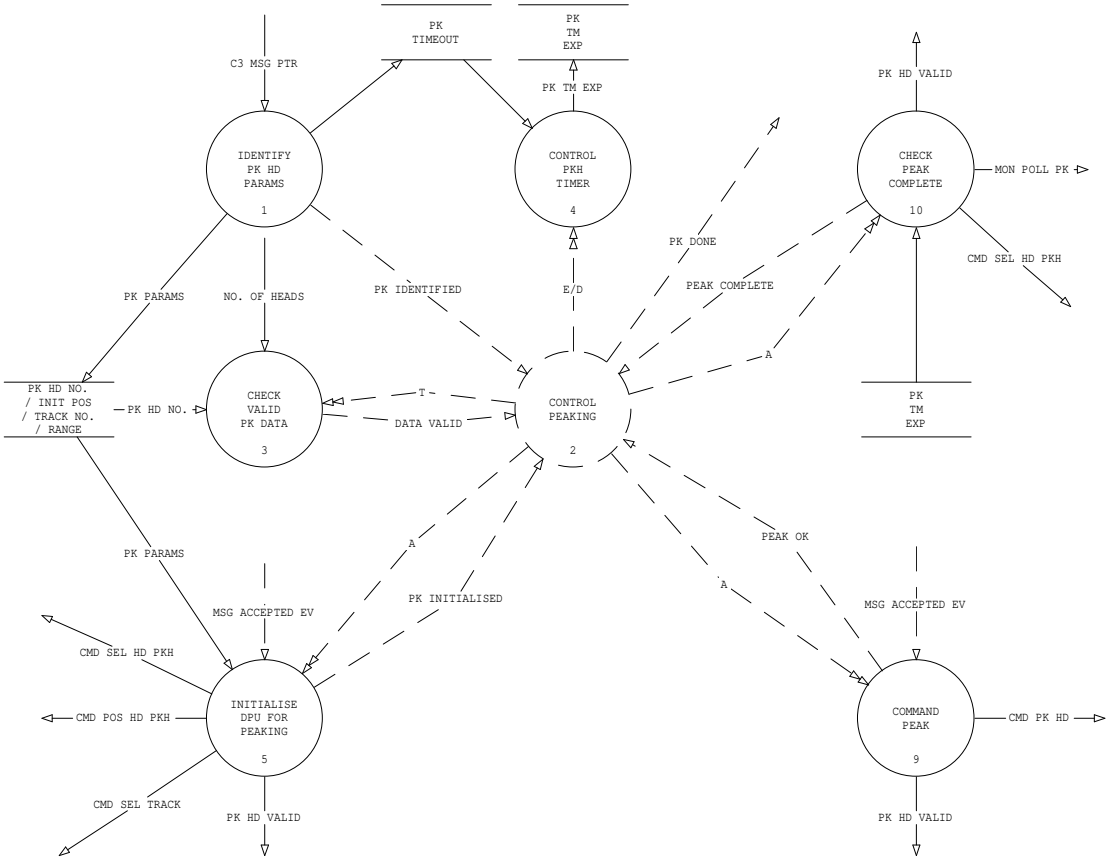
CONTROL POS HEAD CMDS.



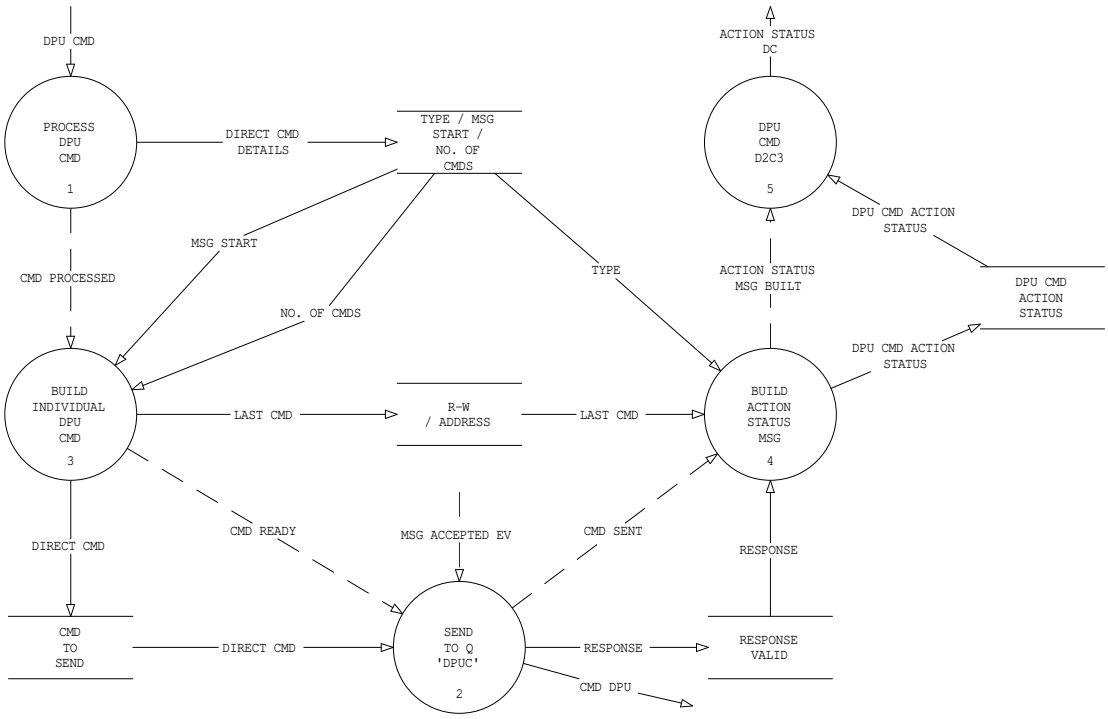
DO POSITION HEAD.



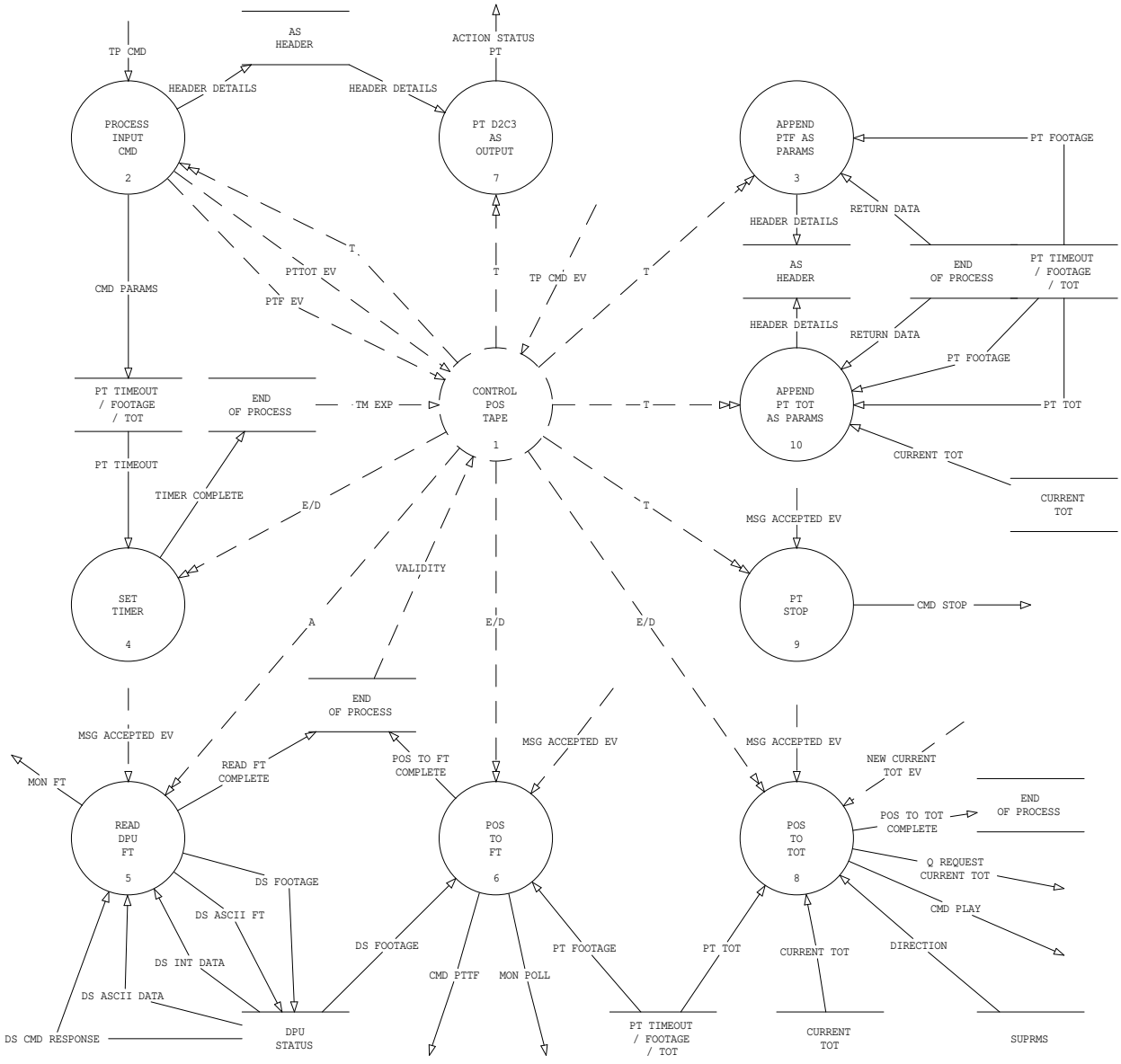
DO PEAK HEADS.



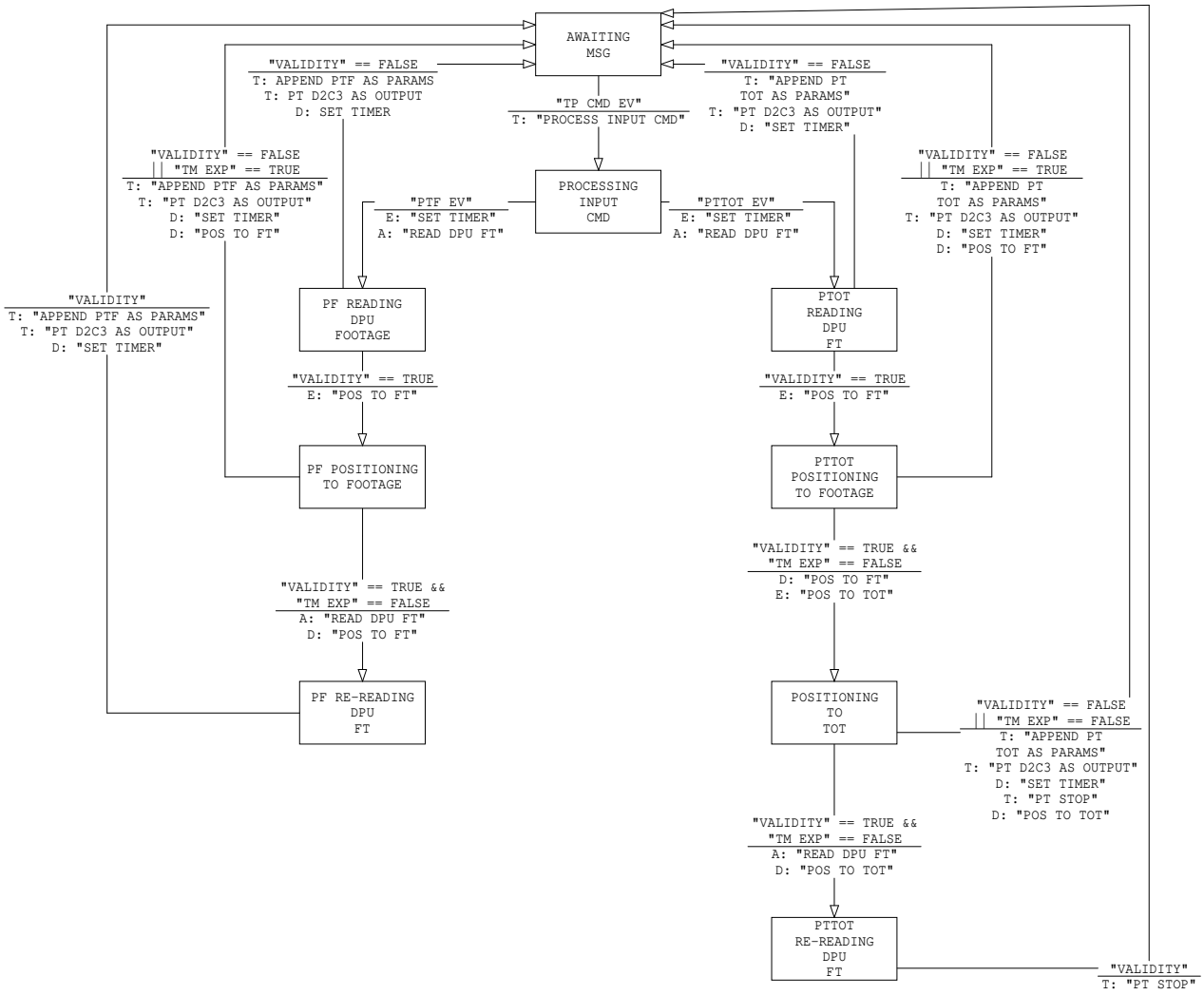
CONTROL DPU CMDS.



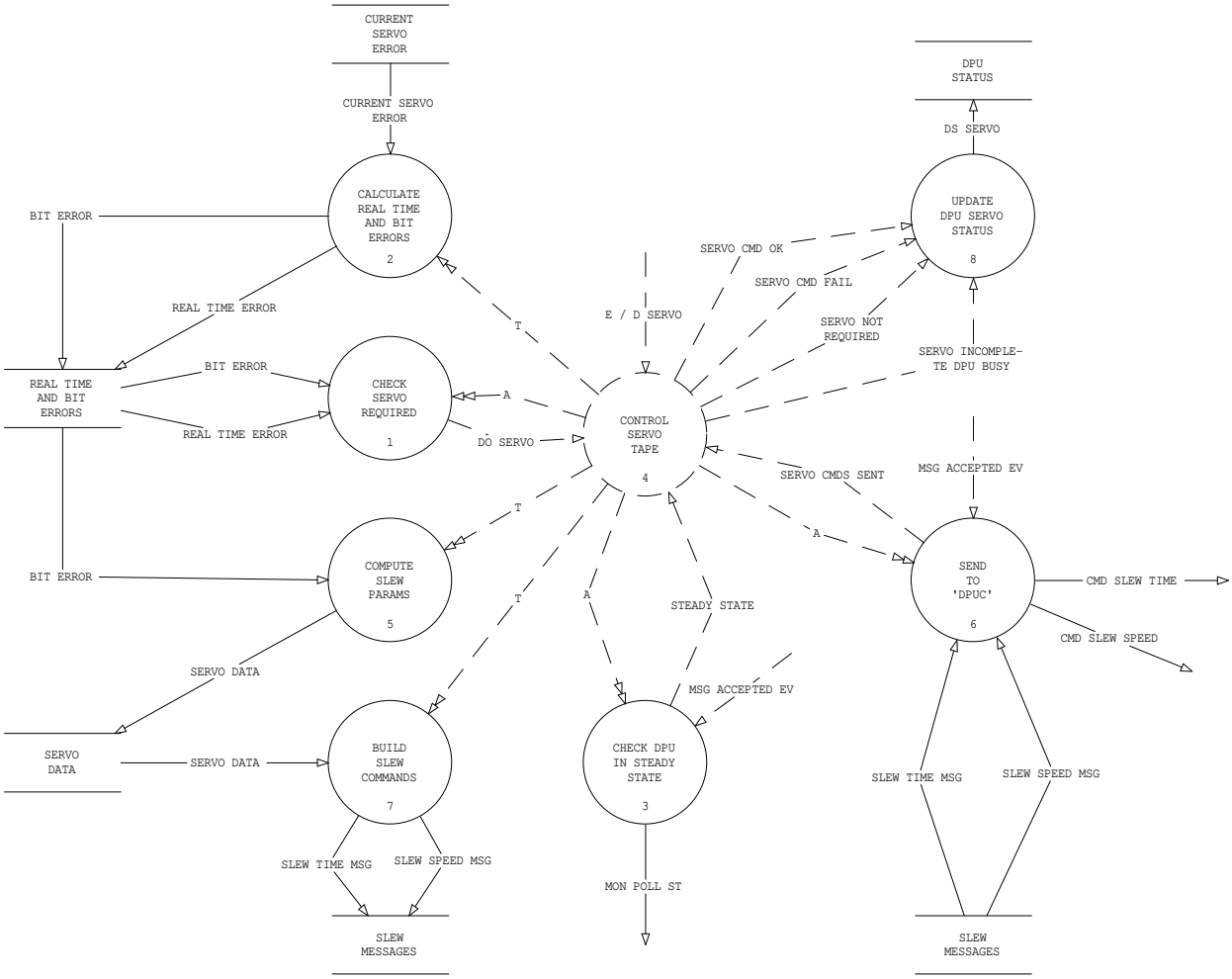
CONTROL POS TAPE CMDS.



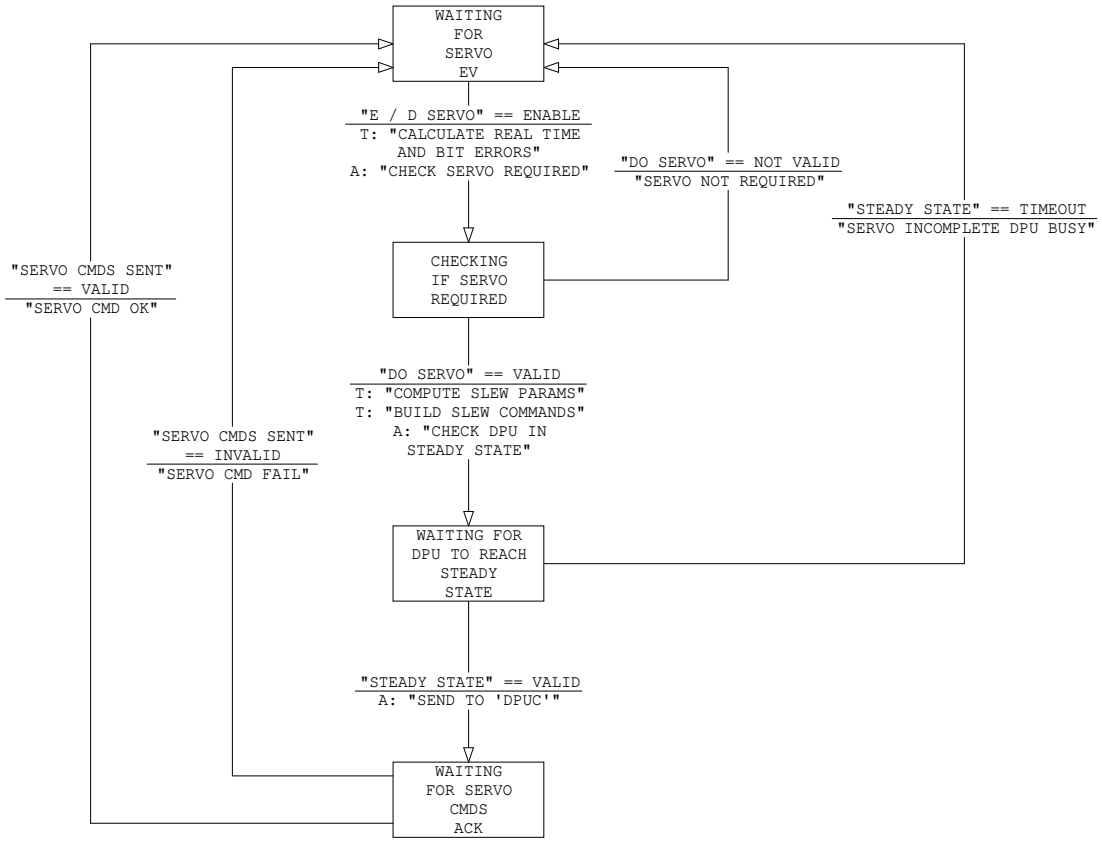
POS TAPE TRANSITION DIAGRAM.



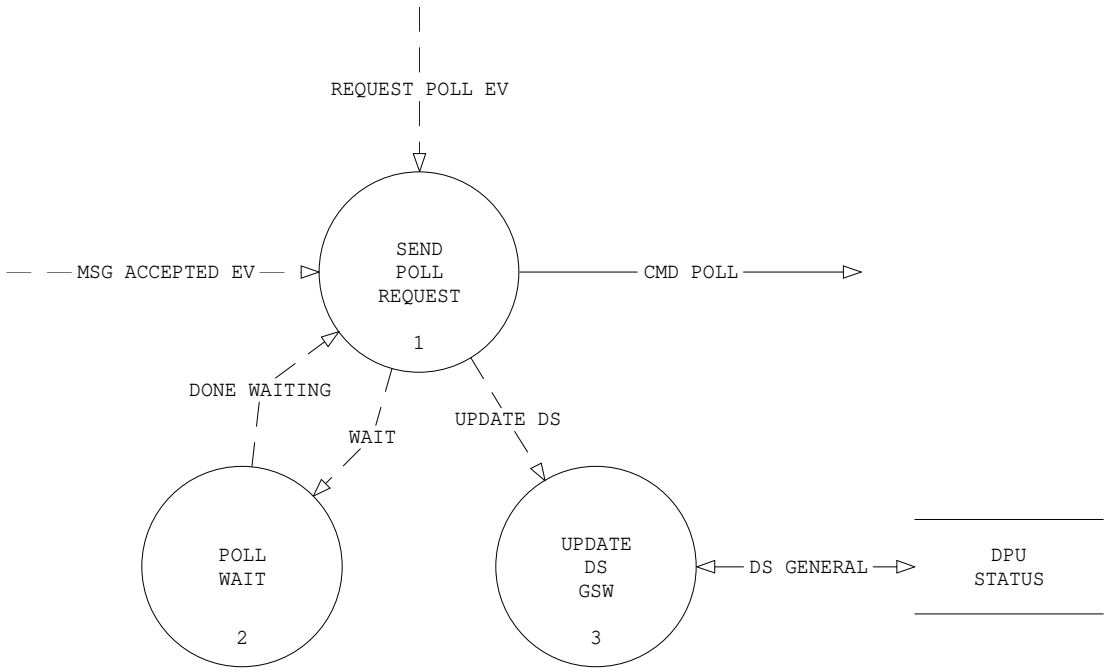
SERVO DPU.


CURRENT SERVO ERROR
CURRENT SERVO ERROR
CALCULATE REAL TIME AND BIT ERRORS
2
BIT ERROR
REAL TIME ERROR
REAL TIME AND BIT ERRORS
BIT ERROR
REAL TIME ERROR
CHECK SERVO REQUIRED
1
T
A
DO SERVO
E / D SERVO
CONTROL SERVO TAPE
4
DPU STATUS
DS SERVO
UPDATE DPU SERVO STATUS
8
SERVO CMD OK
SERVO CMD FAIL
SERVO NOT REQUIRED
SERVO INCOMPLE-TE DPU BUSY
MSG ACCEPTED EV
SERVO CMDS SENT
SEND TO 'DPUC'
6
CMD SLEW TIME
CMD SLEW SPEED
BIT ERROR
COMPUTE SLEW PARAMS
5
T
A
STEADY STATE
SERVO DATA
SERVO DATA
SERVO DATA
BUILD SLEW COMMANDS
7
MSG ACCEPTED EV
CHECK DPU IN STEADY STATE
3
SLEW TIME MSG
SLEW SPEED MSG
SLEW TIME MSG
SLEW SPEED MSG
MON POLL ST
SLEW MESSAGES
SLEW MESSAGES

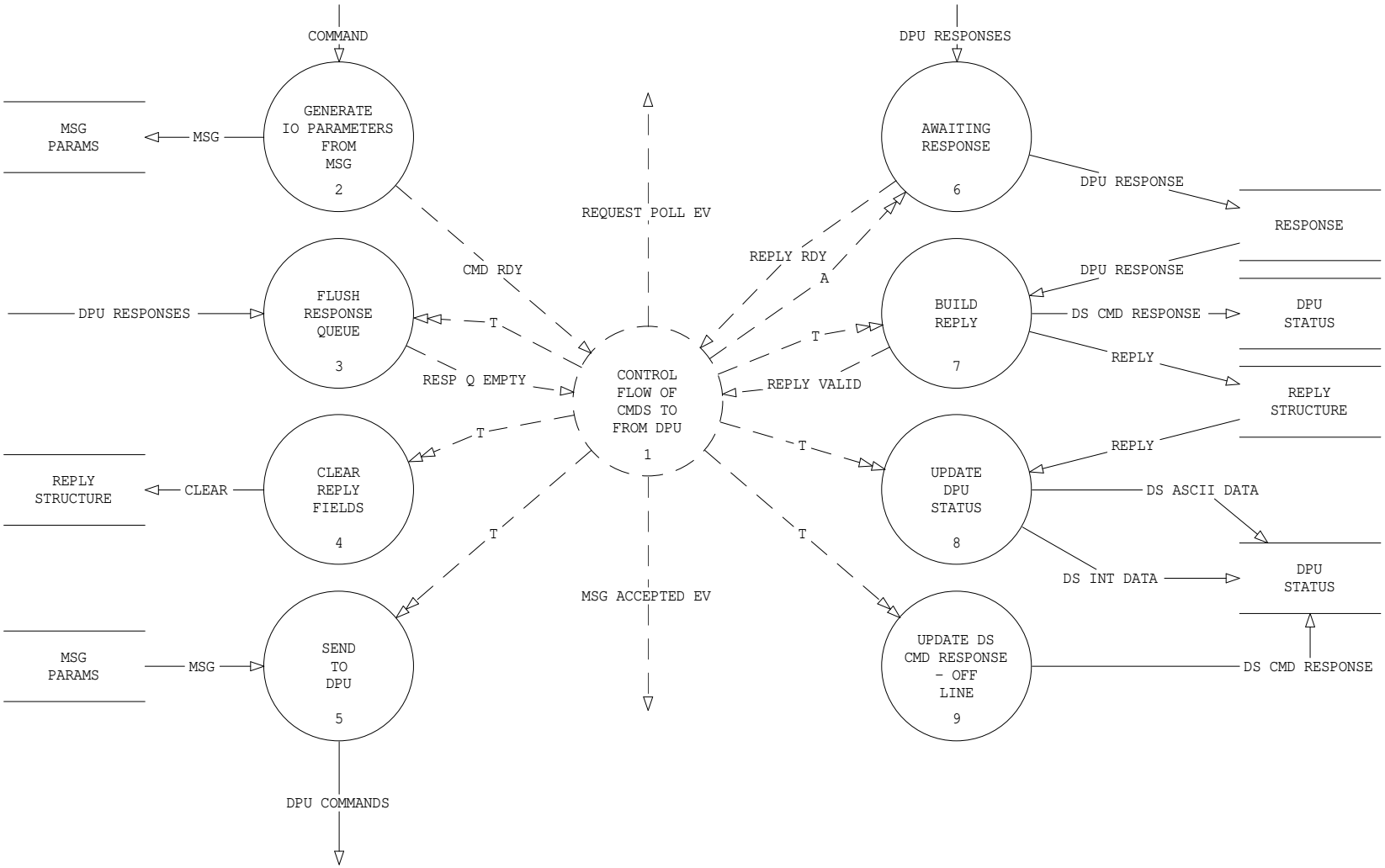
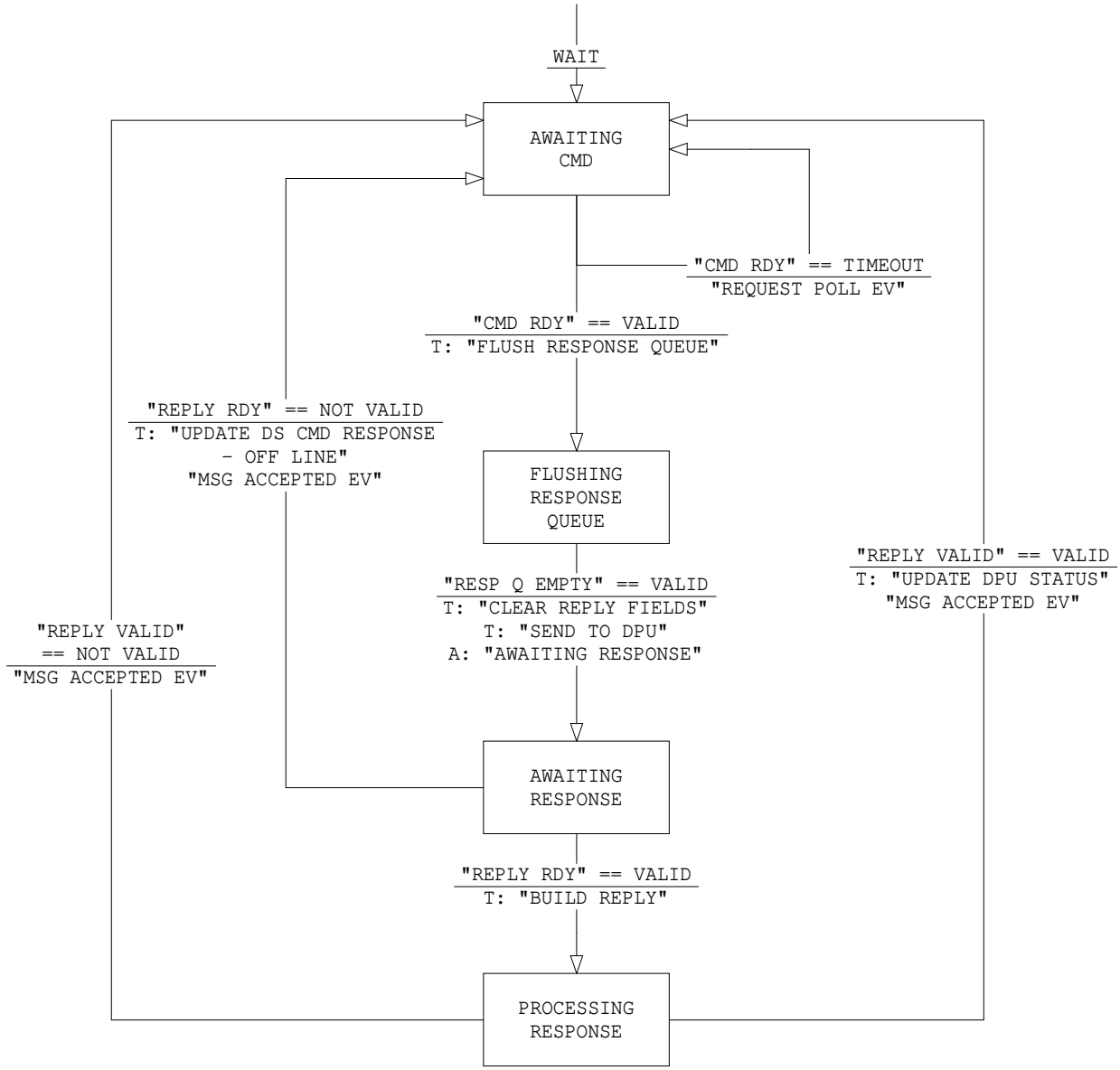
SERVO DPU STATE TRANSISTION DIAGRAM.



POLL DPU.



DEQUEUE DPU COMMANDS.



DEQUEUE DPU COMMANDS STATE TRANSITION DIAGRAM.