PS/CO/Note 91-12
6 juin 1991

The new fast general function generator

(GFAD)

and its software development environment

by Jean Philippe and Maurizio Greco

4 MODES DE FONCTIONNEMENT

Interrupt de changement FL = IRQ6

	0	Autotest	no external timing or strobe
	1	Manlocal	déclenché par le logiciel avant (INIT-EOC) — sans PLS — (1 à 16 fonct.)
non PPM	2	Man Rem EXTSTR	(INIT-EOC-EXTSTR) → 15ns avant INIT — donne parmi les 17 fonctions chargé par l'éditeur
non PPM	3	Man Rem EOCSYNC	même chose — EOC (Synchrone) fait IRQ6
PPM	4	PLS HPLS	mode PPM — (PLS + INIT + EOC)
PPM	5	PLS EXTSTR	comme ci-dessus mais asynchrone.
PPM	6	RTC INT STR	real time avec une fonction A = ayant un vecteur inh. à une fonct B — et un strobe déterminé par une variable.
PPM	7	RTC EXT STR	
PPM	8	PPLS INT STR	généré par le processeur VME —
PPM	9	PPLS EXTSTR	idem

GFAD

Rappel

1) GFA actuel et son environnement
   - problèmes de maintenance
   - boîte noire

2) présentation faite il y a 2 ans
   - software fait par H. LARSEN ; contrôle fait par RS232 avec un Intel)

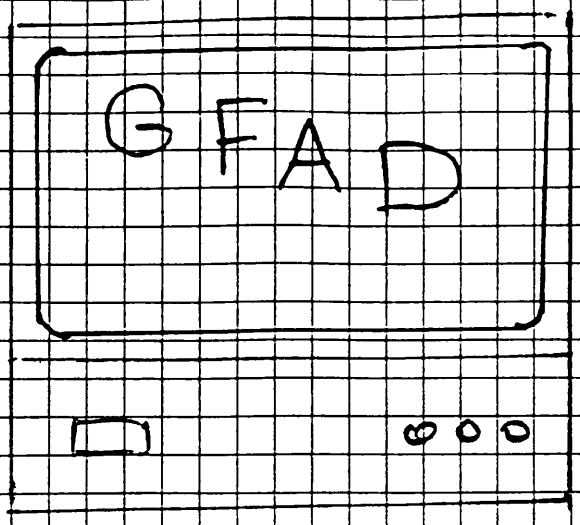
attente
- de l'environnement VME,
- des works Station
- du Reseau ETHernet etc

→ impossibilité de tester complètement le GFAD sans un software complet et facile à manipuler
- édition
- modification avec tests des différents modes de fonctionnement

3) Ce GFA veut couvrir tous les besoins divisionnels.

GFA
- 25% au PS nécessite un Clock Enable au ← 1 µs
- diminuer l'environnement nécessaire pour les Starts. (CAPPC
- fonctions simples ou compliquées
- Générateur de perturbations.
- Echangement des fonctions en cours de fonctionnement
- Facilité de diagnostique et de maintenance (boîte noire

PLS
- PLS actuel
- PLS futur - fait pour le bus VME



- LE "Feuille ton" GFA (1984-1991)
- Son HISTOIRE

1974	sortie " du GFA "OLD"
1984	premieres discussions
1986	specifications pour un nouveau GFA
1988	essai d'un prototype en version VME
1990-91	finalisation et sortie VME du GFAD "NEW"

- Son HISTORIQUE

  - donnera sur plusieurs cycles le secret de la "boite noire".

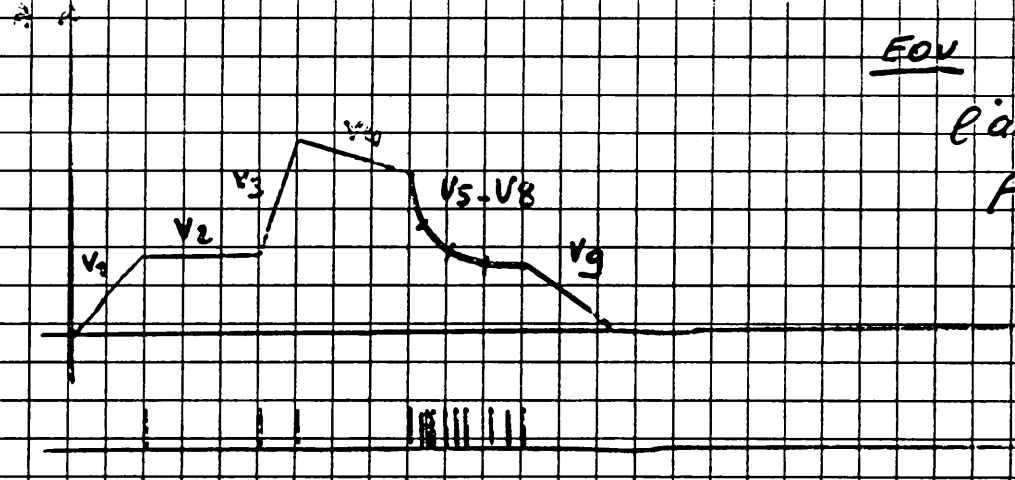
6 DIAGNOSTIQUE ET MAINTENANCE

- Vérification temps réel

Pour chaque fonction, à chaque cycle les données caractérisant une fonction (impulsions, trains, n° fonction, ligne PLS) sont mémorisées.

- Acquisition

Les données digitales de chaque fonction présentes à l'entrée du DAC sont acquises et mémorisées en mode EOV ou Zoom ou mixte.



EOV : valeur de l'amplitude en fin de vecteur

Zoom : Point de départ et freq. échantill.

- Historique

Les données "Temporelles" concernant chaque fonction peuvent être mémorisées pendant 256 cycles en mode FIFO

4 MODES DE FONCTIONNEMENT

4-1 CONTROLE

FL = IRQ6

- AUTOTEST 0
- MAN LOCAL 1
- MAN REM EXT STR 2
- MAN REM EOC DEL 3
- PLS HPLS 4
- PLS EXT STR 5
- RTL INT STR 6
- RTL EXT STR 7
- PPLS INT STR 8
- PPLS EXT STR 9

Tous codés en SIM.

4-2 ACQUISITION

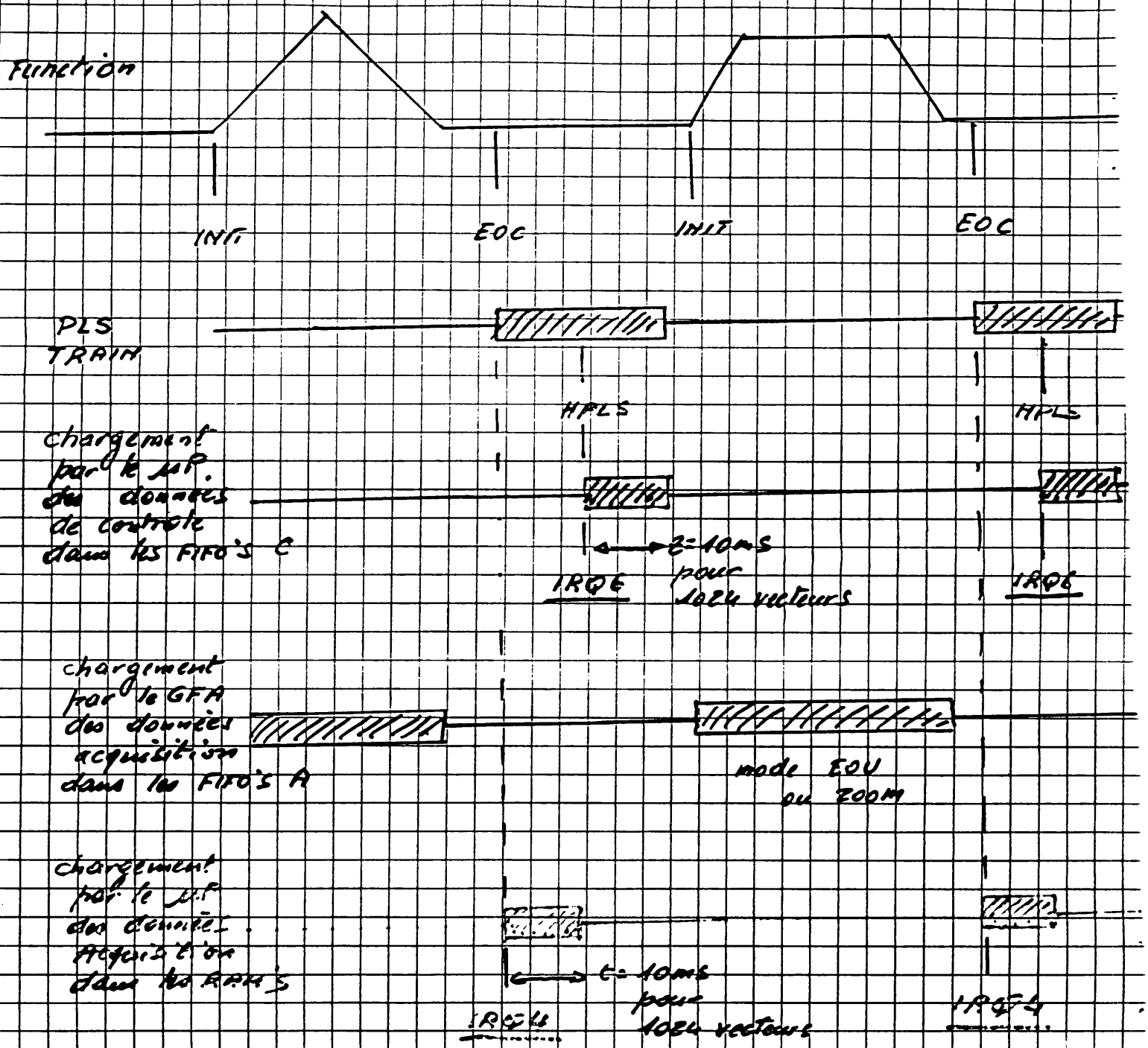
ACQ = IRQ4

- EOV
- ZOOM

5 HORLOGE ET SYNCHRONISATION

- HORLOGE INTERNE 8MHz
- Trains VIF } 250ns, 1.25 µs, 12.5 µs, 125 µs
  VIA } 1.25 ms, EXT.
- Trains EXT et IMPULSIONS synchronisés
- changement de train entre 2 vecteurs : 125 ns.
- Remise à zéro des compteurs entre 2 vecteurs
- Mise à la valeur INITIALE en 2 step de 250ns.

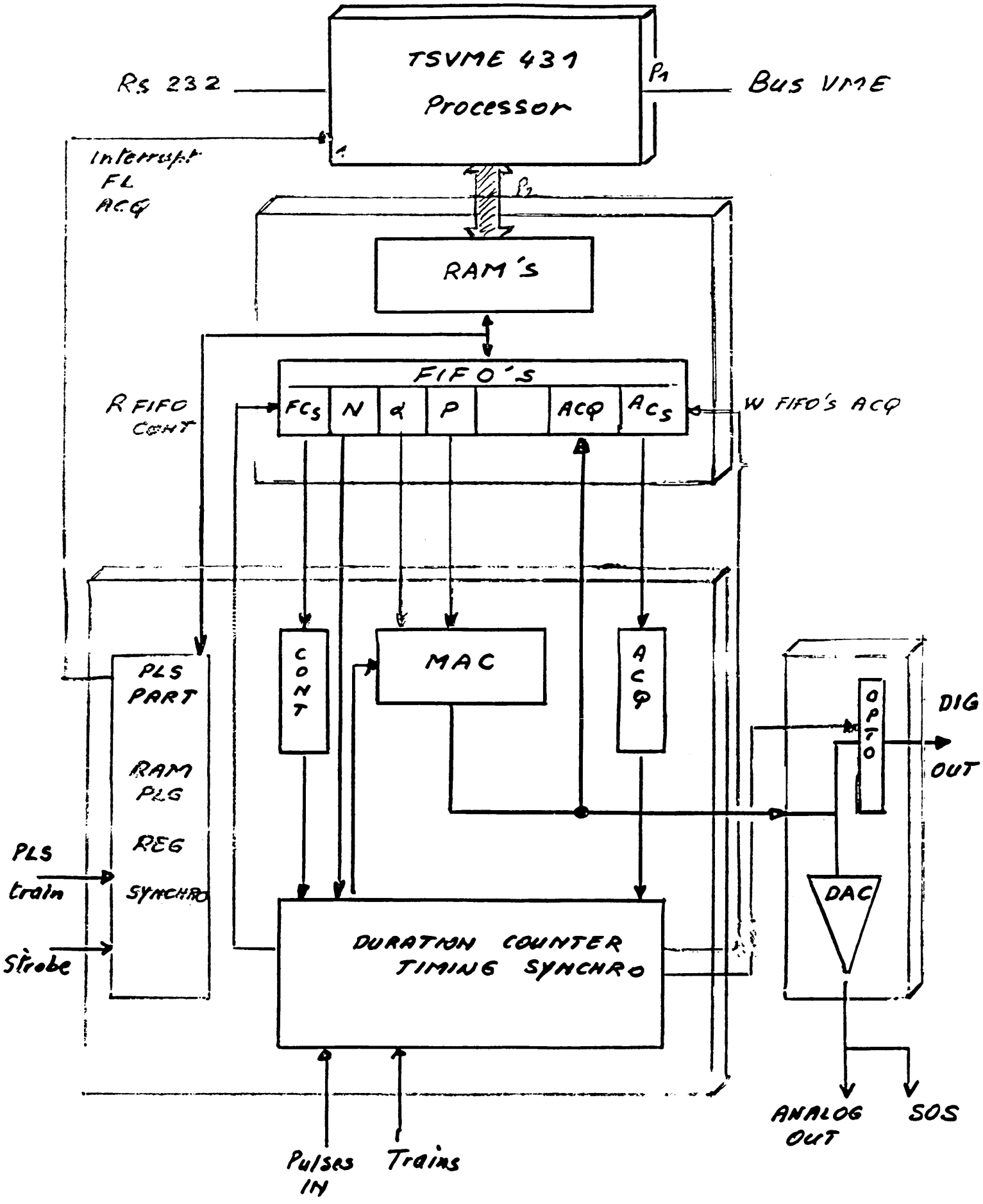
4.3 Diagramme des temps



exemple de temps pour le mode 4 : PLS HPLS

3

SYNOPTIQUE



2 CARACTERISTIQUES GENERALES

- Contrôle par bus VME
- Nombre de fonctions 16 + 1
- Nombre de vecteurs par fonction : 1024

- Nombre d'incréments par vecteur.
  - minimum . 1
  - maximum . 65535
- Durée d'un vecteur.
  - minimum . 250 ns
  - maximum - interne 1,25 ms x
    externe - Ext clock
- Pente d'un vecteur.
  - minimum . ∿ 2^{-16} et ϕ
  - maximum - ∿ 2^{14}
- Sortie analogique.
  - DAC 16 bits ± 10v
- Sortie SOS
  - avec indication de simulation
- Sortie Digitale
  - isolation par optocoupleurs
  - 16 bits + strobe
- INITIALISATION et test du GFA à la mise sous tension
- Possibilités de stop interne (vecteurs longue durée)

1 PRESENTATION DU MODULE

1.1 Mécanique

- module VME 2 unités de large
- carte TSVME 431 de THOMSON (THEMIS)
- Processeur 68010 (10 MHz)
- Contrôle par bus VME
- Carte fille : memoires RAM + FIFO's
- Carte GENE : Traitement PLS et génération
- Carte Digital-analog output

1-2 Impulsions

-
-
- Ext START
- RESET
- PLS Strobe

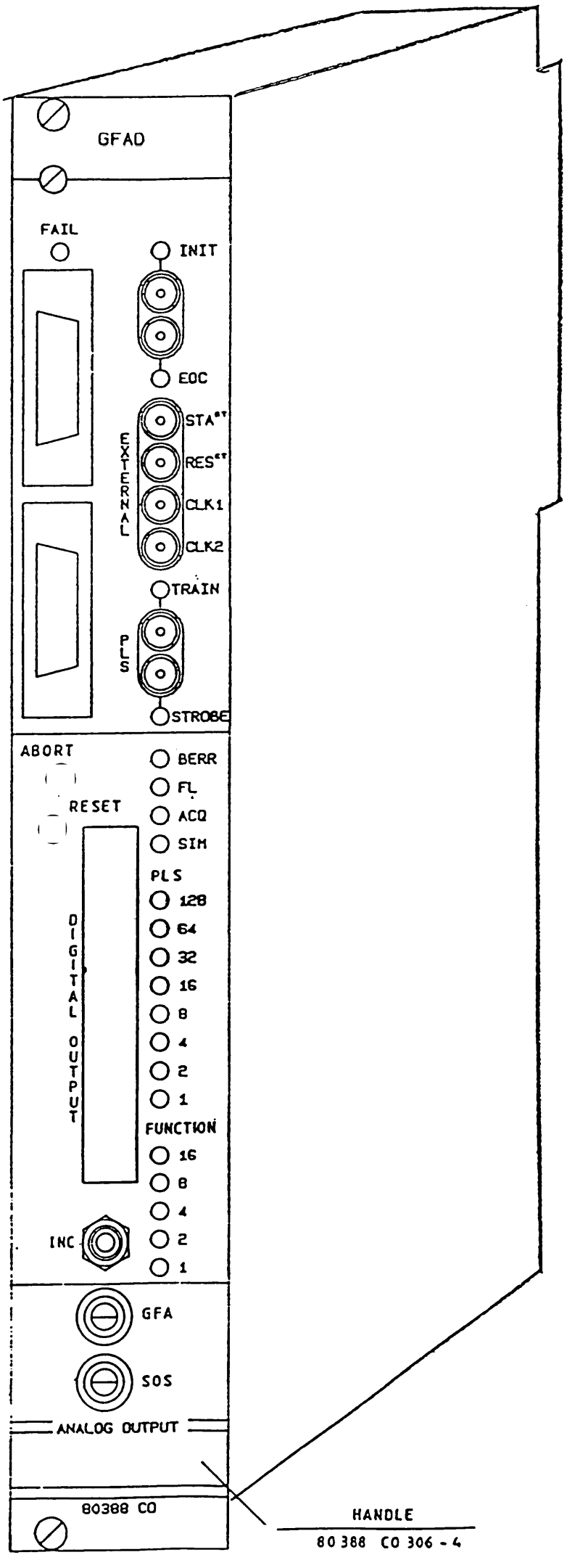
1.

- PLS
- Ext CLOCK 1
- Ext CLOCK 2

} Freq. Max : 4 MHz

1-4 Sorties

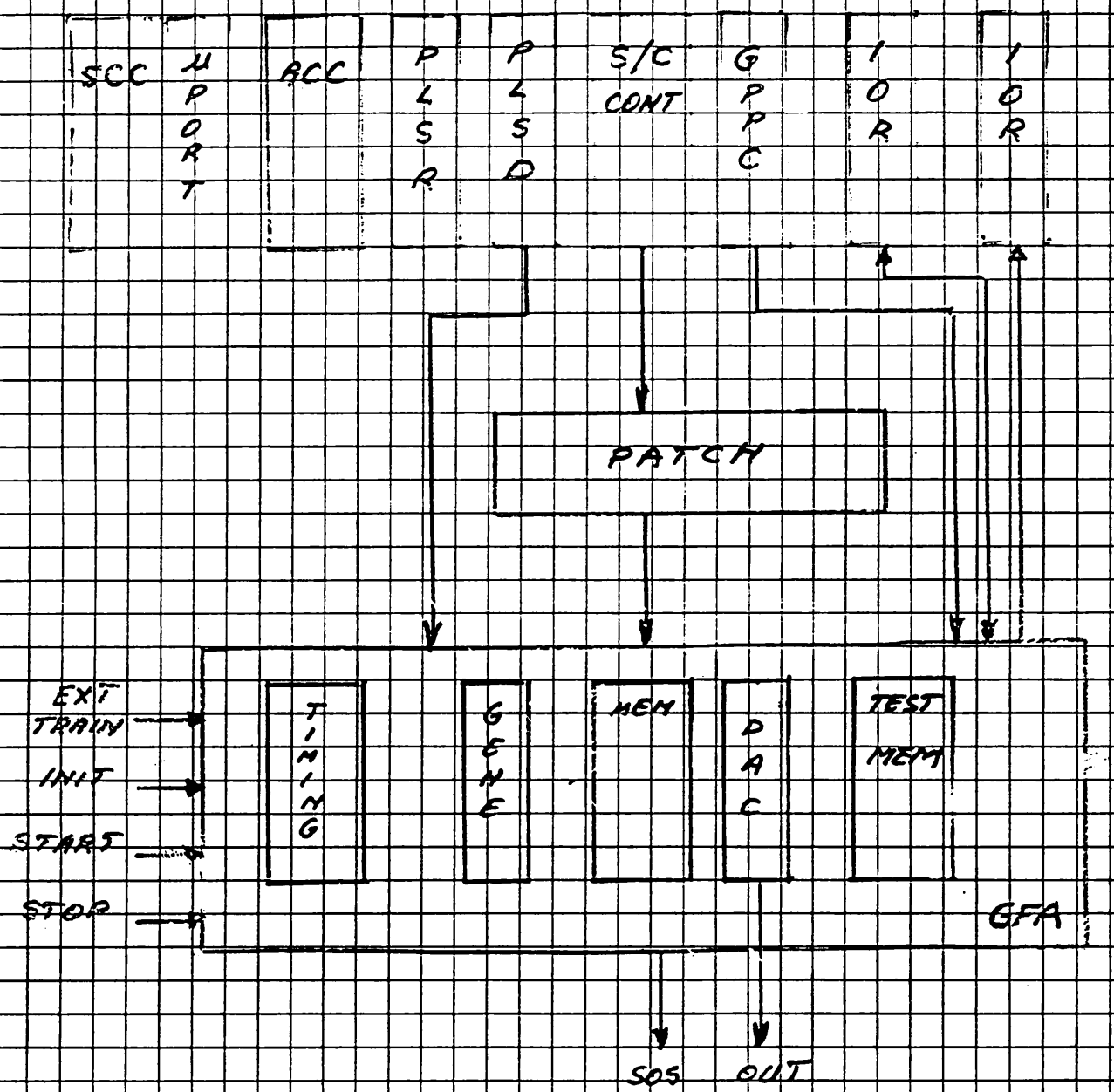
- Analogue (LEMO 2 pins isolée)
- Digitale (16 bits + strobe)
- SOS


GFAD
FAIL
INIT
EOC
EXTERNAL
STA**
RES**
CLK1
CLK2
TRAIN
PLS
STROBE
ABORT
RESET
BERR
FL
ACQ
SIM
PLS
128
64
32
16
8
4
2
1
DIGITAL OUTPUT
FUNCTION
16
8
4
2
1
INC
GFA
SOS
ANALOG OUTPUT
80388 CO
HANDLE
80 388 CO 306 - 4

Bus
VME

GFAD

RAPPEL GFA Actuel



7 DIVERS

INTEGRATION

- Carte Generation utilisant des EPLD's (EP 310 , EP 600)
- exemple

PRIX.

environ 3500 SF.

PARTIE "SOFT"

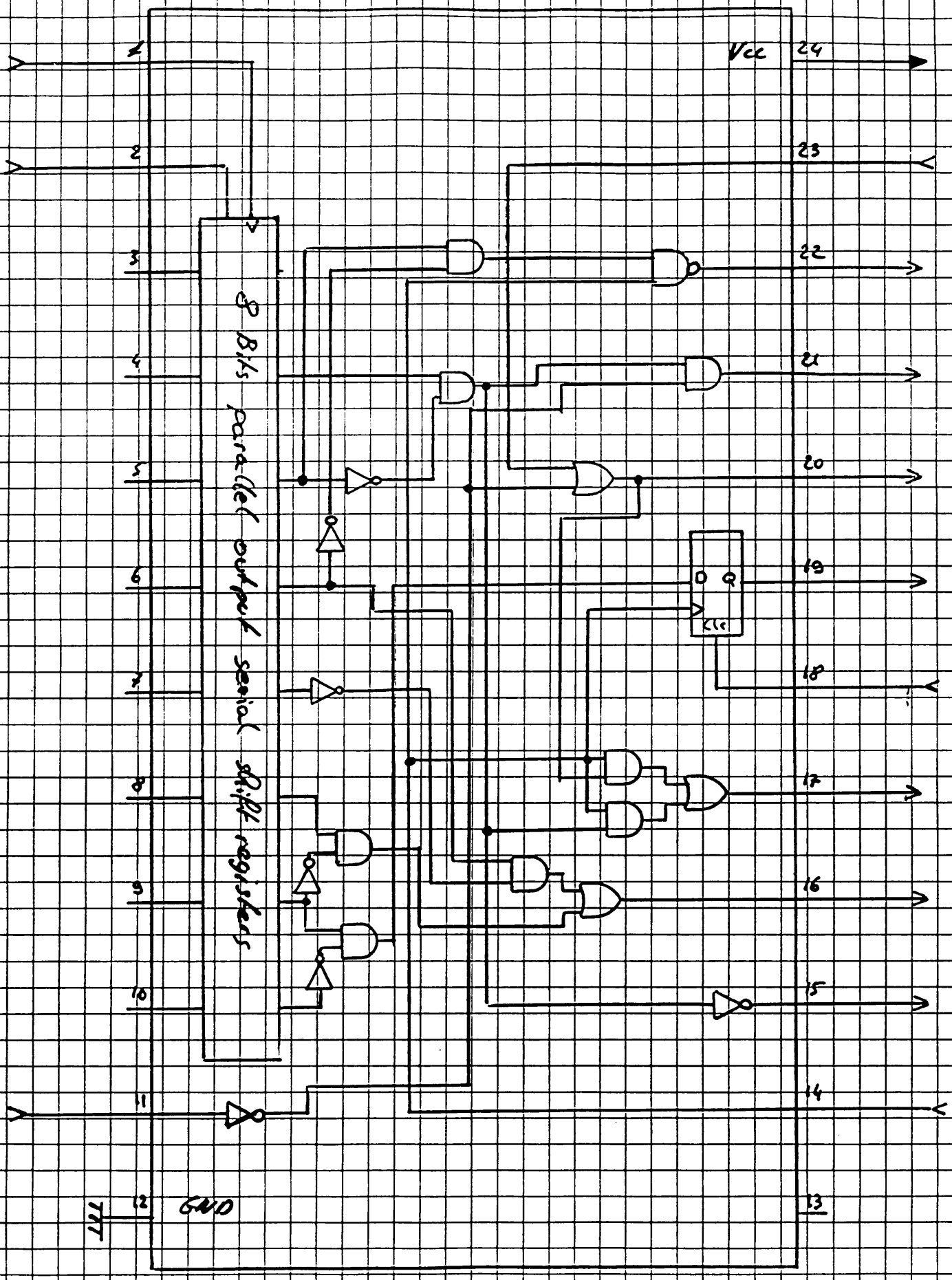
Mauricio GRECO

communication GFA ⟶ CPU VME
environnement THEMIS TSVME 437
de fonction

who? who?

- programmes d'application
- edition
- maintenance et .

EPLD 600


8 bits parallel output serial shift register
Vcc
GND
TTT
D
Q
CLK

GFAD

1 — PRESENTATION DU MODULE
- mécanique
- Impulsions
- Trains
- Sorties

2 — CARACTERISTIQUES GENERALES

3 — SYNOPTIQUE

4 — MODES DE FONCTIONNEMENT
- Contrôle
- Acquisition
- Diagramme des temps.

5 — HORLOGE ET SYNCHRONISATION
- Trains interpes
- Changement de train
- Vecteur INIT

6 — DIAGNOSTIQUE ET MAINTENANCE
- VERIFICATION temps réel
- Acquisition
- Historique

7 — DIVERS
- Intégration
- PRIX
- Partie "SOFT"

GFAD SOFTWARE

1) INSTALL OF OS9 IN TEKMIS CARD TSVME431

2) OS9 MODULE "STARTUP"

   - Architecture
   - Communication via BUS VME
   - Memory Map

3) TEST PROGRAM FOR GFAD

GFAD SOFTWARE DEVELOPMENT

1) HARDWARE ARCHITECTURE

2) DEVELOPMENT SYSTEM SOFTWARE

3) PROGRAM DEBUGGER (FINAL) EPROMs

WHY OS9 ?

This was chosen to avoid to write a specific monitor and driver for the serial port.

It has also given the possibility to write in C the exception routines to manage the interrupts level four and six:

- F$IRQ OS9 System State System Call

  It installs an IRQ service routine into the system polling table.

DEBUGGER facilities.

REMARK: The Debugger is not a symbolic debugger it means that we need to produce also the assembler code generated from the compiler.

```
* ================================================== *
* File /userb/gfad/os9/connect_isr.a
*
* This is intended to provide a way to connect an ISR
* without any device descriptor to a main C program.
* The called ISR will be written in C.

* Take care: the main program has to execute in SV mode
*            and the corresponding system attribute must
*            be set at link-edit

* The called ISR will execute with the system stack, but
* will share its global variables with the calling program
* ================================================== *
        psect   connect_isr,0,0,0,0,0



* ================================================== *
* Provide C callable function to connect an ISR:
* int connect_isr(procfunc,vector)
* int (*procfunc)();
* int vector;
* ================================================== *

connect_isr:
        movem.l  a2-a3,-(a7) save registers

        movea.l d0,a3             : a3 = port address (will carry
        move.l  d1,d0             : d0 = MC680x0 vector number
        moveq   #0,d1             : d1 = prior (we do not handle
        movea.l a6,a2             : a2 = global variables pointer
        lea     isr_entry(PC),a0 :a0 = true ISR entry point
        OS9     F$IRQ
        bcs.s   ce_1
        moveq   #0,d1
ce_1    move.l  d1,d0             : return F$IRQ error as functic
        movem.l (a7)+,a2-a3
        rts

* ================================================== *
* Main ISR entry point
* ================================================== *
* System does gives back the hand here with following registers
* a2 = F$IRQ request a2
* a3 =               a3
* a6 = system global variables pointer
* a7 = system stack pointer (!!!! limited to 1 Kb !!!!!!)
*
isr_entry
        link    a5,#0
        movem.l d0-d7/a0-a6,-(a7)       : save all registers

        movea.l a2,a6                   : call specified routin
        jsr     (a3)

        movem.l (a7)+,d0-d7/a0-a6       : restore all registers
        unlk    a5                      : exit
        rts

        ends
        end
```

PROGRAM RUNNING IN THE GFAD: OS9 MODULE "STARTUP"

C PROGRAM NO MULTITASKING:

- one task in order to minimize
  the knowledge of os9 and the maintenance of
  the application itself (task at level 0)

- basically there is:

1) one start_up sequence containing a selftest.

2) one background program doing the communications.

3) one function loading program, connected to the FL.

4) one acquisition program, connected to the FACQ.

The Data is partly in the shared (between GFAD and VME CPU) or in User (only accessible by the GFAD processor) memory.

DATA STRUCTURE

There are 3 mail boxes for communication:

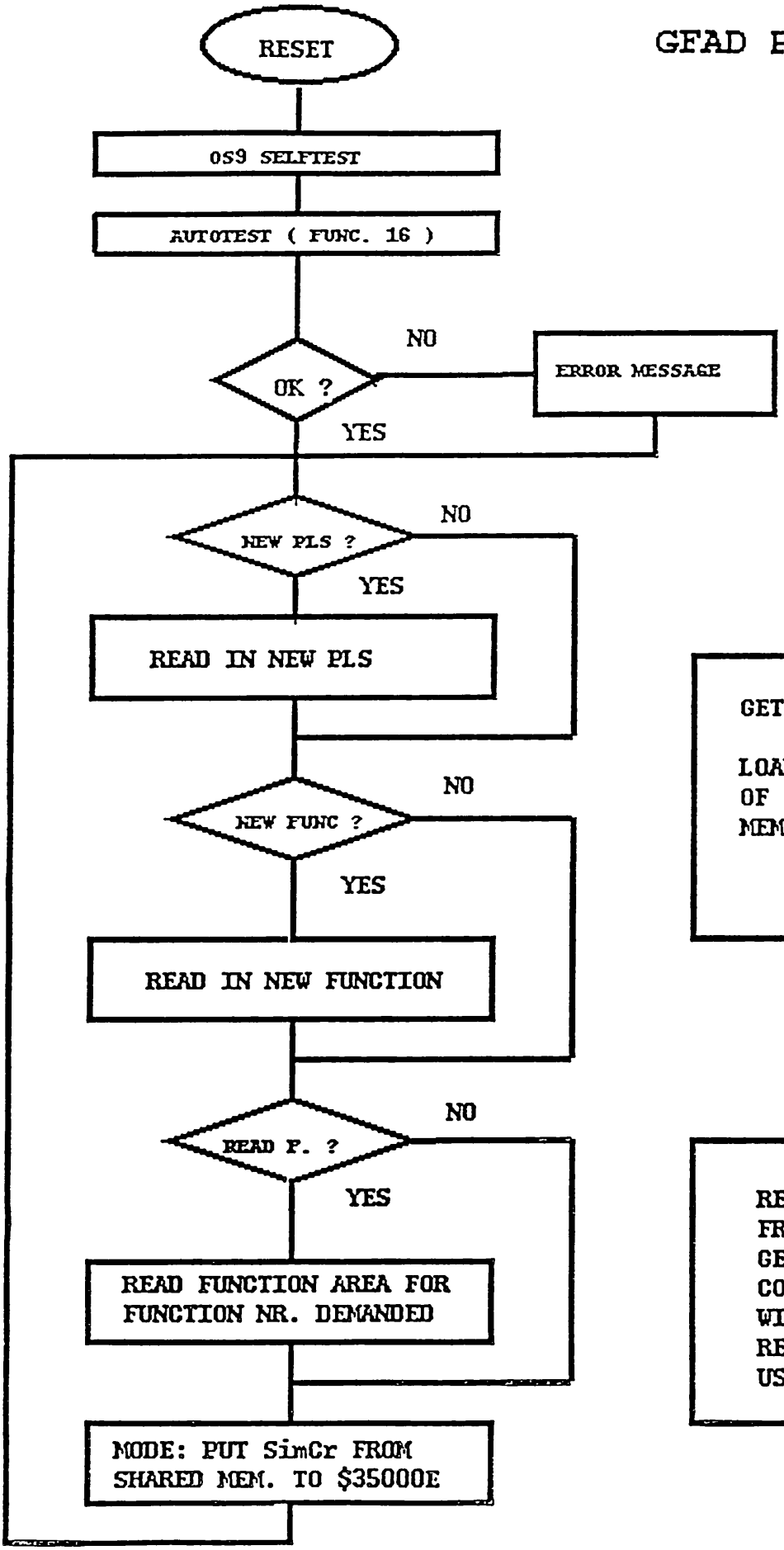
1) function area (to load a new function).

2) acquisition area (to acquire a generated function, containing also error message).

3) PLS RAM area (to load a new PLS pattern).

Other data have fixed addresses in the shared memory, they are accessed by both processors and are not named mailboxes, because the data is not moved to the user memory.

GFAD PROGRAM FLOW CHART



FL (IRQ_6)

GET FUNCTION AREA POINTER

LOAD SLOPE REPRESENTATION OF FUNCTION FROM USER MEMORY INTO FIFOS

return from interrupt

FACQ (IRQ_4)

READ ACQUIRED FUNCTION. FROM FIFO ACQ.
GET FUNCTION AREA POINTER.
COMPARE ACQUIRED FUNCTION WITH THE ONE IN USER RAM.
RESULTS OF COMPARISON INTO USER RAM.

return from interrupt

GFAD ADDRESS MAPPING: FINAL SYSTEM

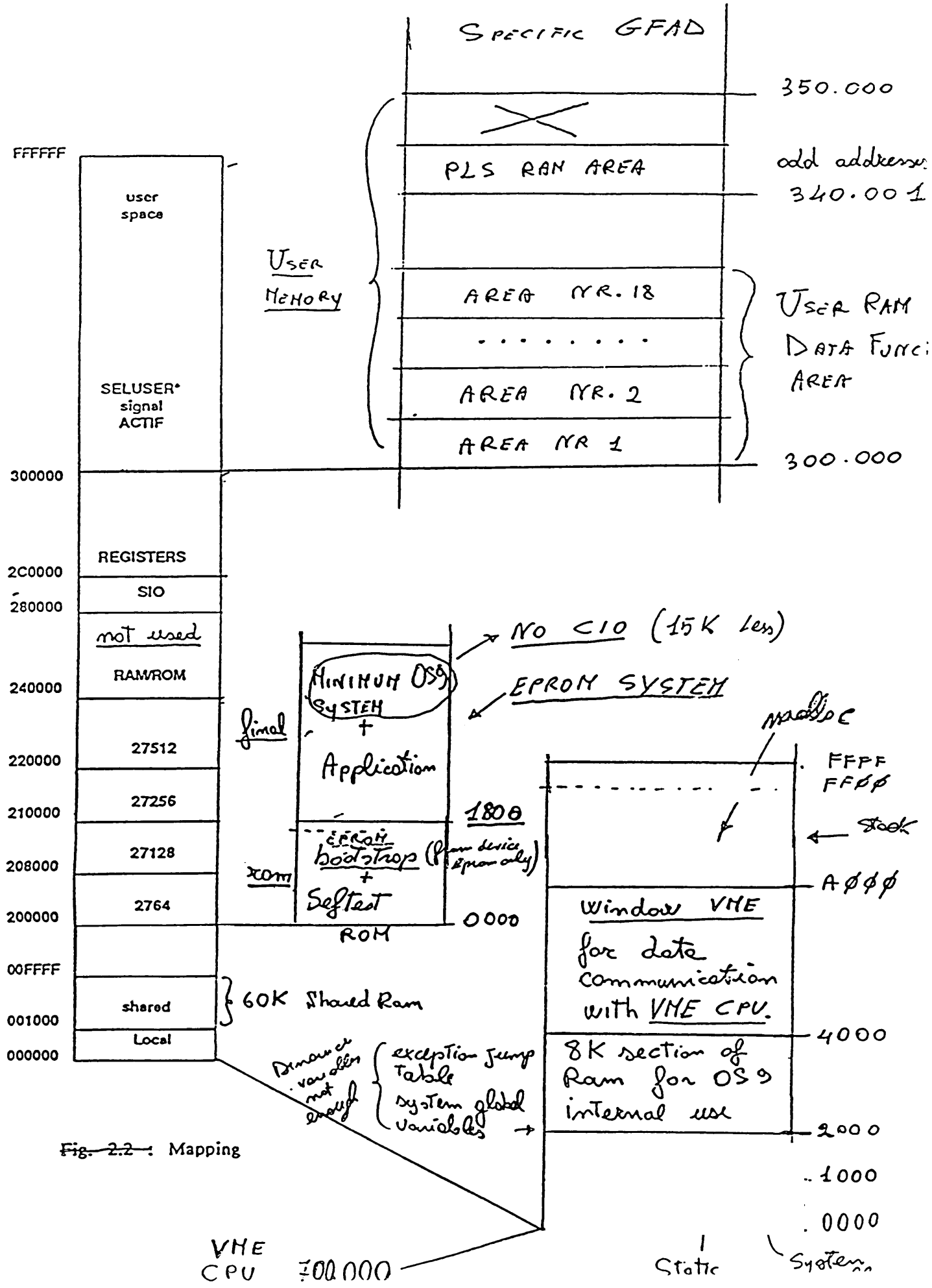


Fig. 2.2: Mapping

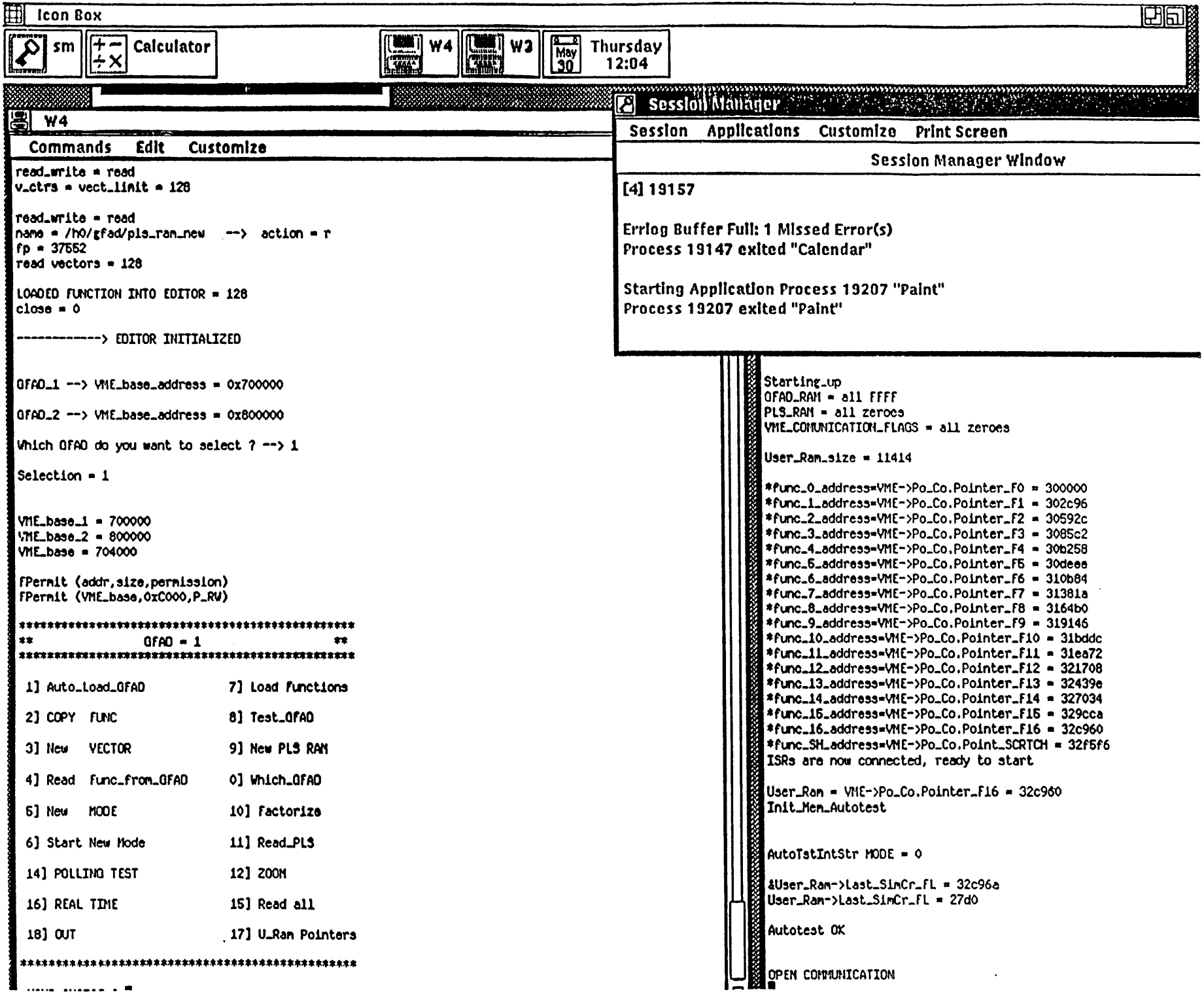
TEST PROGRAM FOR THE GFAD MODULE

- RUN IN OS9 CRATE CONTROLLER

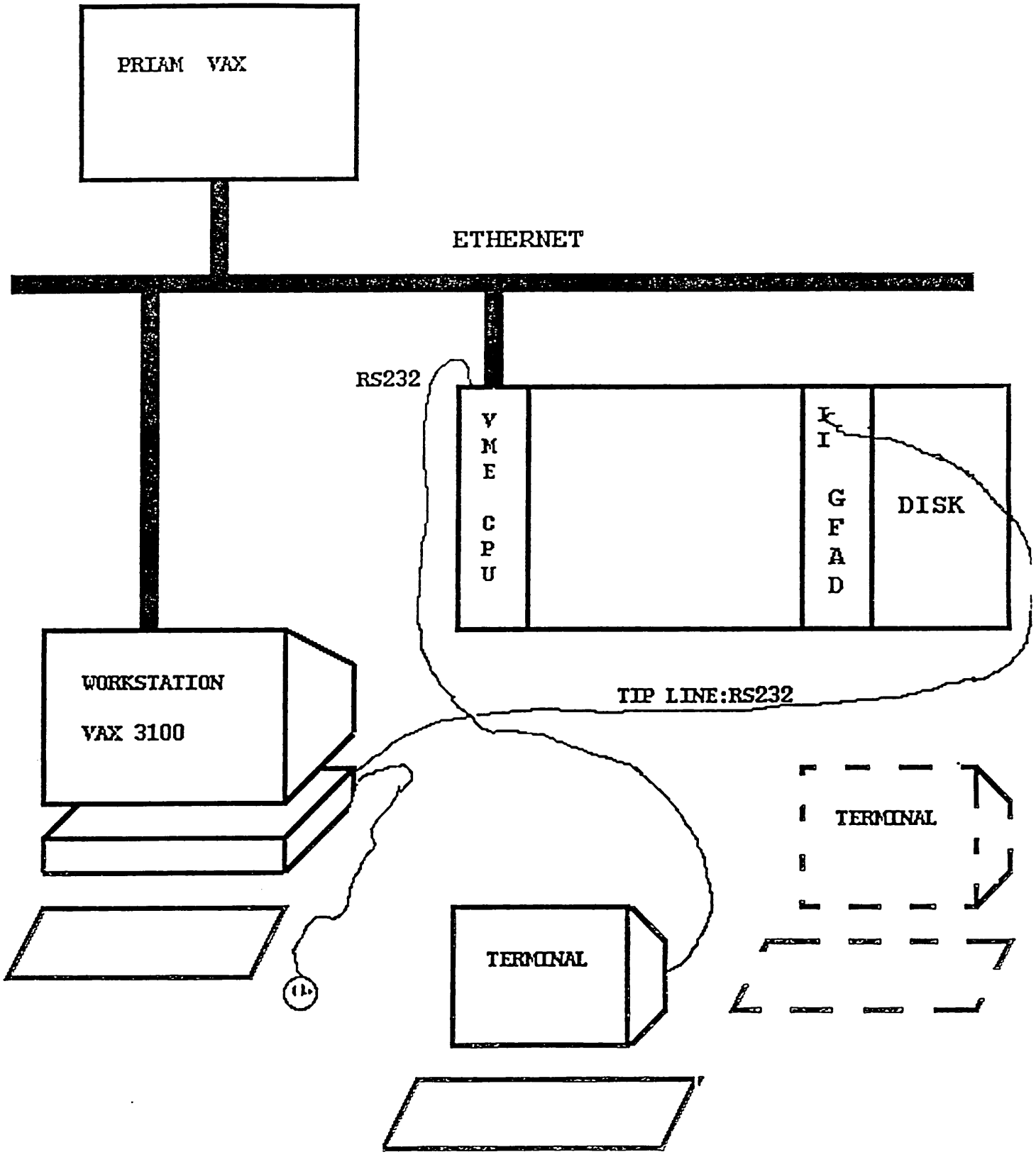
- FOLLOWS A FLAG PROTOCOL TO OPEN THE BUS VME COMMUNICATION WITH THE GFAD MODULE

- IMPLEMENT DIFFERENT FACILITIES TO HELP IN GFAD TEST

- SOME ROUTINES DEVELOPPED, PROBABLY, CAN BE HELPFUL FOR THE NEXT E.M.


Icon Box
sm
Calculator
W4
W3
May 30
Thursday
12:04
W4
Commands Edit Customize
read_write = read
v_ctrs = vect_limit = 128
read_write = read
name = /h0/gfad/pls_ram_new --> action = r
fp = 37552
read vectors = 128
LOADED FUNCTION INTO EDITOR = 128
close = 0
----------> EDITOR INITIALIZED
GFAD_1 --> VME_base_address = 0x700000
GFAD_2 --> VME_base_address = 0x800000
Which GFAD do you want to select ? --> 1
Selection = 1
VME_base_1 = 700000
VME_base_2 = 800000
VME_base = 704000
fPermit (addr,size,permission)
fPermit (VME_base,0xC000,P_RW)
** GFAD = 1 **
1] Auto_Load_GFAD
2] COPY FUNC
3] New VECTOR
4] Read func_from_GFAD
5] New MODE
6] Start New Mode
14] POLLING TEST
16] REAL TIME
18] OUT
7] Load Functions
8] Test_GFAD
9] New PLS RAM
0] Which_GFAD
10] Factorize
11] Read_PLS
12] ZOOM
15] Read all
17] U_Ram Pointers
Session Manager
Session Applications Customize Print Screen
Session Manager Window
[4] 19157
Errlog Buffer Full: 1 Missed Error(s)
Process 19147 exited "Calendar"
Starting Application Process 19207 "Paint"
Process 19207 exited "Paint"
Starting_up
GFAD_RAM = all FFFF
PLS_RAM = all zeroes
VME_COMUNICATION_FLAGS = all zeroes
User_Ram_size = 11414
*func_0_address=VME->Po_Co.Pointer_F0 = 300000
*func_1_address=VME->Po_Co.Pointer_F1 = 302c96
*func_2_address=VME->Po_Co.Pointer_F2 = 30592c
*func_3_address=VME->Po_Co.Pointer_F3 = 3085c2
*func_4_address=VME->Po_Co.Pointer_F4 = 30b258
*func_5_address=VME->Po_Co.Pointer_F5 = 30deee
*func_6_address=VME->Po_Co.Pointer_f6 = 310b84
*func_7_address=VME->Po_Co.Pointer_F7 = 31381a
*func_8_address=VME->Po_Co.Pointer_F8 = 3164b0
*func_9_address=VME->Po_Co.Pointer_F9 = 319146
*func_10_address=VME->Po_Co.Pointer_F10 = 31bddc
*func_11_address=VME->Po_Co.Pointer_F11 = 31ea72
*func_12_address=VME->Po_Co.Pointer_F12 = 321708
*func_13_address=VME->Po_Co.Pointer_F13 = 32439e
*func_14_address=VME->Po_Co.Pointer_F14 = 327034
*func_15_address=VME->Po_Co.Pointer_F15 = 329cca
*func_16_address=VME->Po_Co.Pointer_F16 = 32c960
*func_SH_address=VME->Po_Co.Point_SCRTCH = 32f5f6
ISRs are now connected, ready to start
User_Ram = VME->Po_Co.Pointer_F16 = 32c960
Init_Mem_Autotest
AutoTstIntStr MODE = 0
&User_Ram->Last_SimCr_FL = 32c96a
User_Ram->Last_SimCr_FL = 27d0
Autotest OK
OPEN COMMUNICATION

GFAD SOFTWARE DEVELOPMENT ENVIRONMENT



GFAD DEVELOPMENT SYSTEM SOFTWARE

1) CROSS-SOFTWARE DEVELOPMENT IN VAX: Priam or VaxStation 3100

- cc68 and 168 ---> module "startup"
- Modification of TSVME431/OS9 Software distribution (Makefiles)
- Develop of a few utilities:
  - recompile
  - a working fixmod required to force module owner = 0
- Install a romsplit utility (it produces binary)

2) ftp TRANSFER TO CRATE CONTROLLER DISK

todsc - shell script for transfering files to DSC computer

3) USE THEMIS "DLOAD" UTILITY TO WRITE CODE (startup) INTO SHARED MEMORY AND SIGNAL END OF LOAD.

- rsh dscps003 dload -a=70ff00 -d ./final/startup
- systype.d → Debugger.cc

GFAD ADDRESS MAPPING: DEVELOPMENT SYSTEM

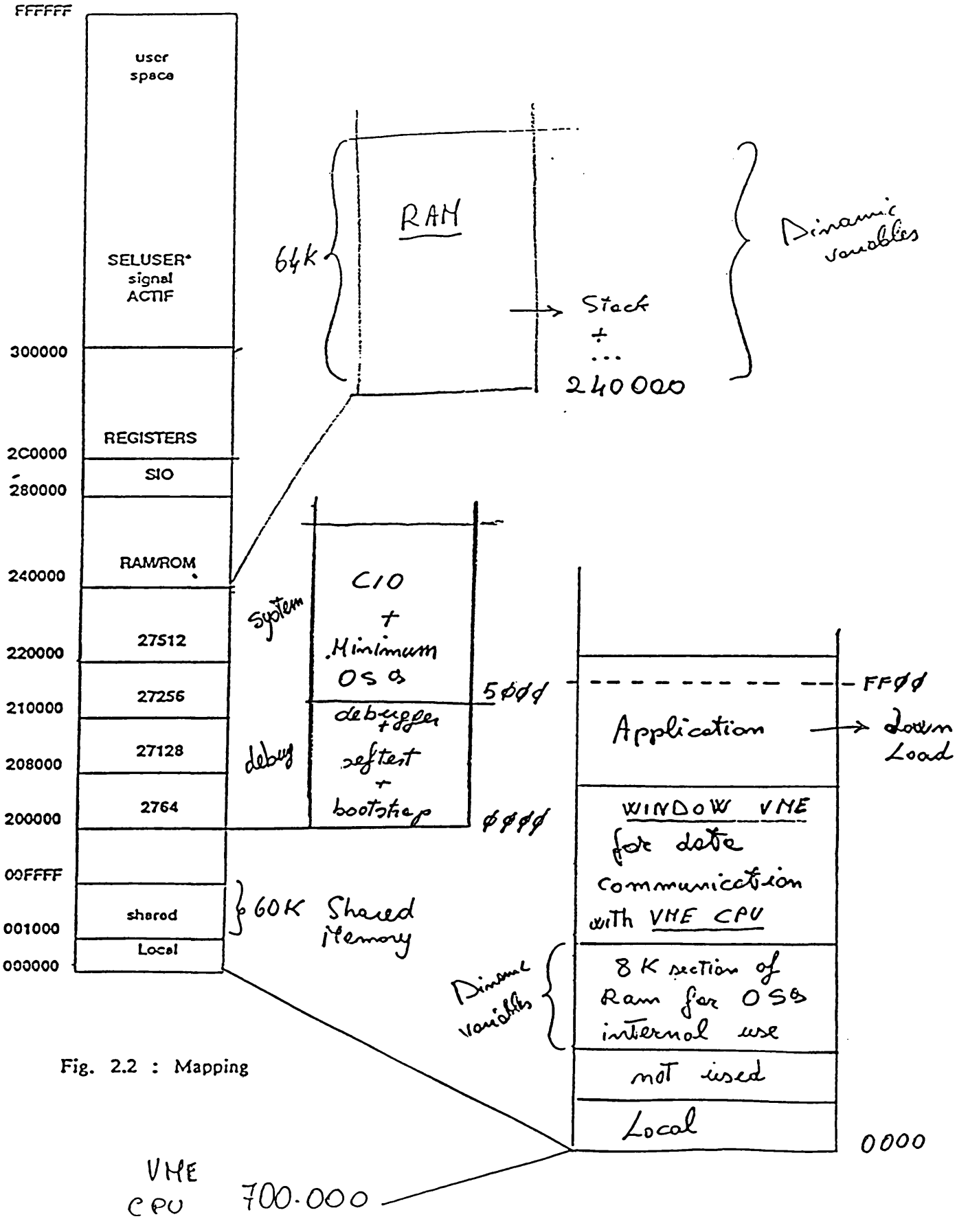
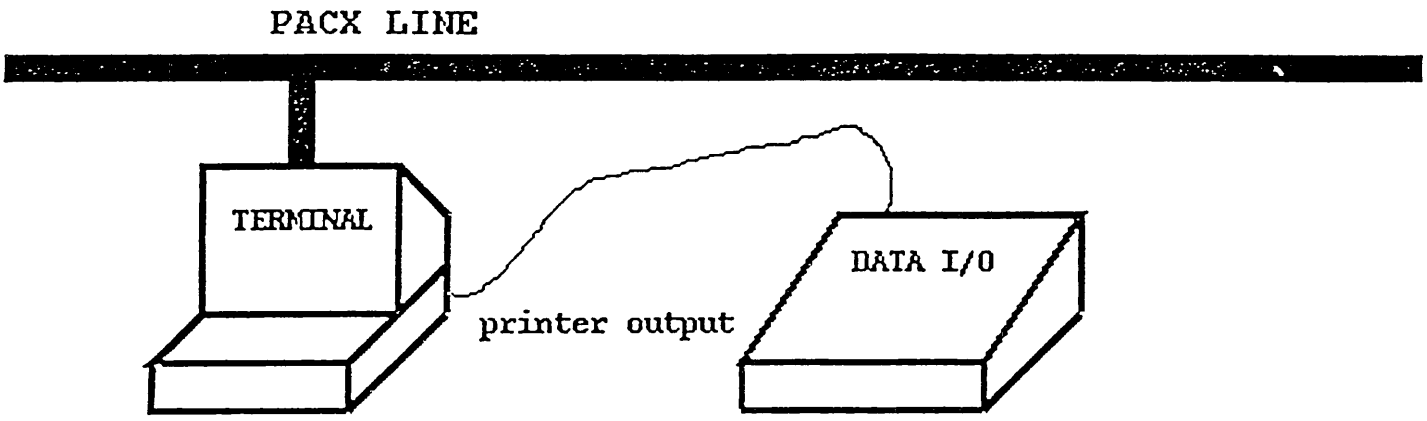


Fig. 2.2 : Mapping

EPROM GENERATION



1) MODIFY "systype.d" in /userb/gfad/T*/T*/DEFS

2) recompile in ~/T*/T*

3) C PRIAM VIA PACX LINE

```
binex debug.0 (rom.0)     --> $0000    even
binex debug.1 (rom.1)     --> $0000    odd

binex system.0 (final.0) --> $2800 ($C00) even
binex system.1 (final.1) --> $2800 ($C00) odd
```

II - PROCEDURE FOR APPLICATION DEVELOPMENT USING TSOFT1430 SOFTWARE

It is assumed that the user is in possession of a development system for writing the programs to govern his application. The development system should run with OS9 version 2.2, and be equipped with at least one serial port and one floppy drive. The development of an application involves the following stages:

/1/ Set up various hardware elements
(see chapter III)

/2/ Set up development system software
(see chapter III)

/3/ Program debugger EPROMs for installation on target system
(see chapter IV)

/4/ Generate application (containing minimum OS9 software) on development system (see chapters V and VI)

/5/ Load application on target system and debug
(see chapter VI)

/WHILE/ application fails to work correctly
/THEN/ repeat stages 4 and 5
/END WHILE/

/6/ Generate final system
(see chapter VII)

III - SETTING UP

3.1 - HARDWARE ARCHITECTURE REQUIRED

The required hardware architecture is shown below:

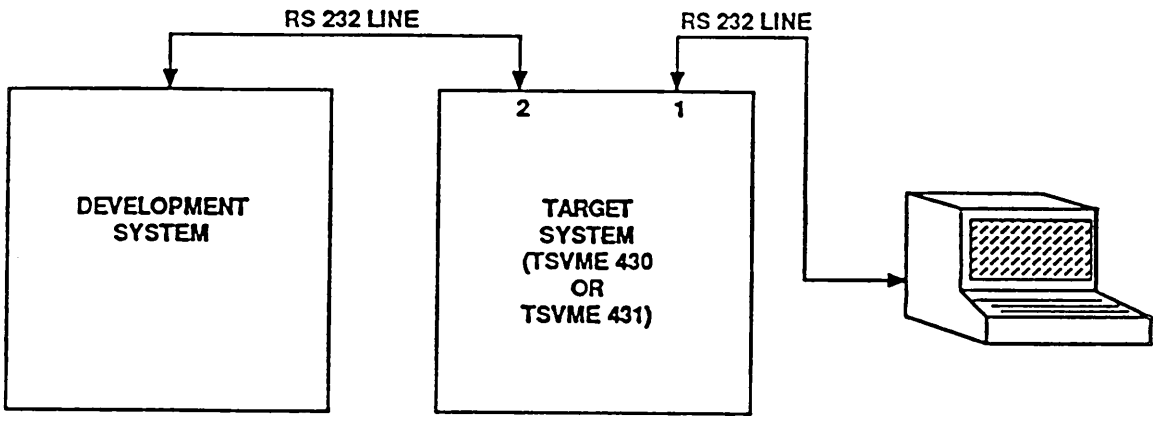


Figure 3.1 : Hardware architecture required for application development

A special debugger command (tm) switches the target system over to transparent mode, upon which the terminal will appear to be linked straight through to the development system.

This means that the same terminal can be used for dialogues with the development system or the target system. Programs are transferred between the two machines via the serial link.