Architecture d'un système d'acquisition-Multiprocesseurs

Hervé Postec

▶ **To cite this version:**

Hervé Postec. Architecture d'un système d'acquisition-Multiprocesseurs. Instrumentations et Détecteurs [physics.ins-det]. Université de Caen, 1987. Français. NNT : . tel-02056625

HAL Id: tel-02056625
https://hal.in2p3.fr/tel-02056625

Submitted on 4 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GANIL T.87-03

UNIVERSITE DE CAEN

THESE

présentée

pour obtenir

le DIPLOME DE DOCTEUR-INGENIEUR
Spécialité Instrumentation et Mesures

par

Hervé POSTEC
Ingénieur ISMRA

sujet

— Architecture d'un système d'acquisition — Multiprocesseurs

Soutenue le 10 juillet 1987 devant la commission d'examen :

M. BLOYET	Président
Mme ADAM A. M. BIZARD G. M. DELAGRANGE H. M. RAINE B.	Examinateurs

Thèse présentée à

L'UNIVERSITE DE CAEN

DIPLOME DE DOCTEUR-INGENIEUR

Spécialité Instrumentation et Mesures

- Architecture d'un système d'acquisition - multiprocesseurs

par Hervé POSTEC ingénieur ISMRA

Thèse soutenue le 10 juillet 1987 devant : M. BLOYET D (président)
Mme. ADAM A
M. BIZARD G
M. DELAGRANGE H
M. RAINE B

REMERCIEMENTS

Je remercie en premier lieu Monsieur H. DELAGRANGE d'avoir accepté de diriger cette thèse.

Je remercie particulièrement Messieurs J. CLEMENT et B. RAINE qui m'ont guidé tout au long de cette étude.

Je remercie Messieurs M. TRIPON et J. TILLIER pour les informations qu'ils ont pu me donner, concernant respectivement les modules analogiques et les modules logiques utilisés pour l'acquisition des données au Ganil.

Je remercie Messieurs D. VAILLANT et J.P. LE BLAY pour leurs conseils et leur collaboration dans l'analyse des logiciels du système d'acquisition.

Je remercie Monsieur G. BIZARD pour sa collaboration et ses remarques en tant que futur utilisateur.

Je remercie Monsieur G. RIST pour son aide technique dans les tests que j'ai pu effectuer.

Je remercie aussi toutes les personnes qui de près ou de loin ont contribué à ce travail.

Je remercie Madame A. ADAM d'avoir bien voulu se pencher sur ce document et Monsieur D. BLOYET d'avoir accepté la présidence du jury.

TABLE DES MATIERES

INTRODUCTION (p. 11)

CHAPITRE 1 : Présentation générale du système d'acquisition existant au GANIL (p. 13)

1.1. Le déroulement d'une expérience (p. 13)

1.1.1. Réaction nucléaire et détection des fragments résultants (p. 13)

1.1.2. Orientation vers les dispositifs multidétecteurs (p. 15)

1.1.3. Acquisition des données et dépouillement. (p. 17)

1.2. Le système d'acquisition actuel (p. 18)

1.2.1. Description générale (p. 18)

1.2.2. Déclenchement (p. 20)

1.2.3. Codage (p. 22)

1.2.4. Le microprocesseur CAB (p. 23)

1.2.5. Le rôle de l'ordinateur (p. 24)

1.3. Définition d'un cahier des charges pour le système d'acquisition des multidétecteurs (p. 25)

1.3.1. Caractéristiques générales du système (p. 25)

1.3.2. Caractéristiques de codage (p. 27)

CHAPITRE 2 : Les bases du système d'acquisition des multidétecteurs (p. 29)

2.1. Le système d'acquisition actuel est-il adapté aux multidétecteurs ? (p. 29)

2.2. Les principes généraux de conception du système d'acquisition (p. 30)

2.2.1. Parallélisme (p. 30)

2.2.2. Utilisation de files d'attente (p. 32)

2.2.3. Filtrage en ligne (p. 33)

2.3. L'organisation des modules CAMAC (p. 35)

2.3.1. Présentation générale (p. 35)

2.3.2. Les codeurs (p. 36)

2.3.3. Les cartes DC (p. 40)

2.3.4. Les problèmes liés à la lecture des voies additionnelles (p. 44)

CHAPITRE 3 : Détermination d'une architecture matérielle (p. 45)

3.1. Choix des microprocesseurs (p. 45)

3.2.1. La famille 68000 (p. 45)

3.2.2. Le bus VME et ses extensions (p. 48)

3.2. Un exemple d'architecture possible (p. 51)

3.2.1. Description générale (p. 51)

3.2.2. Acquisition-Filtrage (p. 51)

3.2.3. L'ensemble Groupement-Transfert (p. 54)

3.2.4. Les problèmes posés par ce type d'organisation (p. 55)

3.3. L'architecture finalement retenue (p. 56)

3.3.1. Description générale (p. 56)

3.3.2. L'interfaçage FERA-VME (p. 58)

3.3.3. La lecture des voies CAMAC (p. 63)

3.3.4. Filtrage des événements (p. 64)

3.3.5. Transfert vers l'ordinateur (p. 65)

3.3.6. Contrôle et visualisation (p. 65)

CHAPITRE 4 : Les logiciels du système d'acquisition (p. 67)

4.1. Choix d'un système d'exploitation et de langage de programmation (p. 67)

4.1.1. Le système d'exploitation (p. 67)

4.1.2. Communication entre le système de développement et les unités centrales cibles (p. 69)

4.2. Lancement d'une application (p. 70)

4.2.1 Système de description d'une application (p. 70)

4.2.2 Communication entre le système de développement et les unités centrales cibles (p. 77)

4.2.3. Logiciel de gestion de mémoire (p. 81)

4.2.3.1. Intérêt (p. 81)

4.2.3.2. Les découpages de la mémoire (p. 81)

4.2.3.3. Mécanismes d'utilisation du logiciel (p. 86)

4.2.4. Chronologie des opérations de démarrage d'une application (p. 88)

4.3. Synchronisation des processus de filtrage (p. 90)

4.3.1. Présentation générale du problème (p. 90)

4.3.2. Synchronisation par arbitrage du bus VME (p. 91)

4.3.3. Algorithme de synchronisation retenu (p. 95)

- CONCLUSION (p. 100)

- REFERENCES BIBLIOGRAPHIQUES (p. 102)

- BIBLIOGRAPHIE (p. 105)

- ANNEXE 1 (p. 106)

- ANNEXE 2 (p. 122)

TABLE DES FIGURES

Fig 1.1 Représentation schématique de la détection des fragments résultant d'une réaction nucléaire. (p. 14)

Fig 1.2 Implantation des multidétecteurs dans NAUTILUS. (p. 16)

Fig 1.3 Schéma simplifié de l'acquisition GANIL. (p. 19)

Fig 1.4 Schéma d'un exemple de déclenchement. (p. 21)

Fig 2.1.a Architecture parallèle et chronogramme associé. (p. 31)

Fig 2.1.b Organisation producteur/consommateur asynchrone et chronogramme associé. (p. 31)

Fig 2.2 Evolution du flux de données et du type de filtrage effectué en allant de l'amont vers l'aval du système. (p. 34)

Fig 2.3 Schéma général de l'organisation des modules CAMAC. (p. 35)

Fig 2.4 Schéma de cablage et de fonctionnement du bus FERA. (p. 38)

Fig 2.5 Format de sortie des données au standard FERA. (p. 38)

Fig 2.6 Organisation des modules de codage pour un multidétecteurs. (p. 40)

Fig 2.7 Décision prise par la carte DC associée au MUR en fonction de la valeur prise par le couple (Q,T) issu de chacun des détecteurs touchés. (p. 42)

Fig 2.8 Organigramme de fonctionnement de la carte DC associée au MUR de PLASTIQUE. (p. 42)

Fig 2.9 Centralisation des décisions prises par les cartes DC. (p.43)

Fig 3.1 La liaison VMX permet de décharger le bus VME d'un grand nombre de transferts. (p. 49)

Fig 3.2 Exemple d'architecture VMV multibranches. (p. 50)

Fig 3.3 Utilisation de l'interface VME/CAMAC. (p.50)

Fig 3.4 Schéma de principe d'une architecture possible. (p. 52)

Fig 3.5 Structure en file d'attente circulaire de la carte ST. (p. 53)

Fig 3.6 Regroupement des sous-évènements par l'ensemble GT. (p. 55)

Fig 3.7 Architecture retenue pour le système d'acquisition des multidétecteurs. (p. 57)

Fig 3.8 Schéma d'implantation des modules VME dans les différents châssis. (p. 59)

Fig 3.9.a Structure générale du format "sous-évènement". (p. 60)

Fig 3.9.b Structuration des données pour chaque type de détecteur. (p. 60)

Fig 3.10 Schéma de fonctionnement de la carte RS. (p. 61)

Fig 3.11 *Avec deux mémoires fonctionnant en bascule (chrono 2), on peut éviter des blocages de l'acquisition qui se produiraient si l'on n'avait qu'un seul tampon en entrée (chrono 1).* (p. 62)

Fig 4.1 Exemple de fichier de description d'application. (p. 72)

Fig 4.2 Organisation des fichiers de description de la carte ASCU2 de FORCE COMPUTERS et des composants qu'elle utilise. (p.74-75)

Fig 4.3 Exemple de fichier de lancement d'application. (p. 76)

Fig 4.4 Usage de la commande TM de PDOS. (p. 77)

Fig 4.5 Utilisation d'une RAM DISK commune à deux châssis pour le téléchargement et l'exécution des logiciels. (p. 90)

Fig 4.6 Structure d'un en-tête de bloc. (p. 82)

Fig 4.7.a Allocation avant. (p. 84)

Fig 4.7.b Allocation chaînée. (p. 84)

Fig 4.7.c Allocation par blocs. (p. 84)

Fig 4.7.d Exemple d'allocations de blocs. (p. 85)

Fig 4.7.e Découpage de la mémoire correspondant aux allocations décrites à la fig 4.7.d. (p. 85)

Fig 4.8 Organisation du chaînage "historique" des attachement d'un processus aux différents blocs de mémoire. (p. 87)

Fig 4.9 Chronologie des opération de lancement d'une application. (p. 89)

Fig 4.10 Synchronisation des processeurs PF par arbitrage du bus VME. (p. 93)

Fig 4.11 Exemple de problème pouvant survenir avec une synchronisation des processeurs n'utilisant que des fonctionnalités matérielles. (p. 94)

Fig 4.12.a Principe de l'algorithme de synchronisation des processeurs PF. (p. 97)

Fig 4.12.b Algorithme de synchronisation des processeurs PF. (p. 98)

INTRODUCTION

En physique nucléaire comme en physique des particules, les expérimentateurs sont amenés, pour caractériser au mieux les réactions qu'ils étudient, à construire des systèmes de détection couvrant de grands angles solides et offrant une bonne résolution spatiale. Il en résulte une augmentation considérable du nombre de paramètres à prendre en compte : le "Château de Cristal" construit par le Centre de Recherches Nucléaires de Strasbourg réunit à lui seul 72 détecteurs. Pour conserver un taux d'acquisition satisfaisant, on doit donc construire des systèmes d'acquisition, eux aussi de plus en plus imposants et présentant d'excellentes performances en rapidité.

Au GANIL, l'implantation de quatre ensembles de détection dans la chambre à réaction NAUTILUS, conduit à des configurations d'expérience dans lesquelles 700 paramètres sont à traiter. Des expériences menées jusqu'à présent comprenant au maximum 250 paramètres ont montré les limitations du système d'acquisition actuel et la nécessité d'étudier un dispositif mieux adapté à la lecture d'un grand nombre de voies.

Notre étude s'est inspirée des résultats obtenus par d'autres laboratoires (Centre de Recherches Nucléaires de Strasbourg, Institut des Sciences Nucléaires de Grenoble). Nous avons en effet retenu, pour notre dispositif d'acquisition, des fonctionnalités déjà mises en place sur d'autres systèmes et des matériels déjà largement utilisés. De plus, nous nous sommes efforcés de mettre en application les principes généraux de conception d'une architecture d'acquisition dégagés par les techniciens du CERN [GAV 85]. Toutefois, les demandes et les contraintes ne sont jamais rigoureusement les mêmes d'un type d'expérience à un autre et il n'était pas possible de "recopier" au GANIL un système d'acquisition déjà existant. Des solutions techniques spécifiques ont donc été déterminées ; ceci d'une part, afin d'utiliser les techniques les plus récentes et d'autre part, pour tenir compte de la structure à quatre ensembles de détection du dispositif.

Le chapitre 1 décrit les différents éléments intervenant dans une expérience de physique nucléaire et permet de situer la place et le rôle du système d'acquisition dans ce contexte. Nous donnerons dans ce chapitre le cahier des charges du système d'acquisition à réaliser.

Nous montrerons dans le chapitre 2 que le système d'acquisition actuel ne répond pas aux critères définis dans le cahier des charges ; ceci notamment du point de vue rapidité. Nous indiquerons alors les lignes directrices de la construction du nouveau système et nous décrirons pour terminer, le front d'acquisition réalisant la conversion analogique/numérique des informations.

Nous décrirons alors dans le chapitre 3, le matériel employé et l'architecture retenue pour effectuer la lecture et le filtrage des données numériques obtenues. Nous donnerons en outre dans ce chapitre, un exemple d'architecture adoptée dans un premier temps puis rejetée du fait de son manque d'optimisation.

Le chapitre 4 présente les différents problèmes logiciels rencontrés et les solutions apportées. L'architecture du système ayant une structure multiprocesseurs, nous détaillerons plus particulièrement l'algorithme de synchronisation des différentes unités centrales de filtrage des données.

CHAPITRE 1

PRESENTATION GENERALE DU SYSTEME D'ACQUISITION EXISTANT AU GANIL

Ce chapitre décrit les différents dispositifs mis en jeu lors d'une expérience de physique nucléaire, en insistant sur le système d'acquisition de données. Il montre, en outre, pourquoi les physiciens s'orientent vers les ensembles multidétecteurs et débouche pour terminer, sur la définition du cahier des charges d'un système d'acquisition adapté à ceux-ci.

1.1. Le déroulement d'une expérience

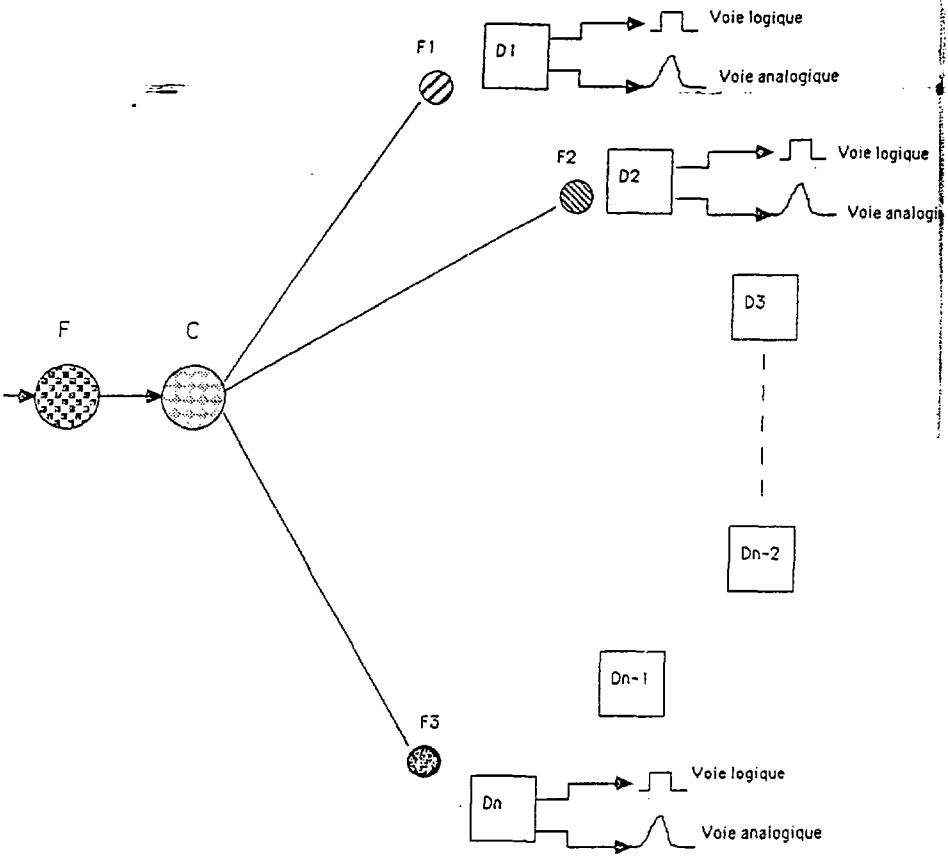
1.1.1. Réaction nucléaire. Détection des fragments

Les expériences menées au GANIL ont pour but, la découverte et l'étude des différents états de la matière nucléaire. Pour ce faire, on provoque des perturbations sur les noyaux d'une cible en exposant cette dernière à un faisceau d'ions lourds. Les réactions nucléaires résultant des différentes collisions faisceau-cible conduisent à l'éjection de fragments divers dont la répartition spatiale et les caractéristiques sont significatives des forces mises en jeu dans le noyau. On cherche donc à détecter et à identifier ces fragments.

Ces deux fonctions sont assurées par les détecteurs disposés [KNO 79, SAM 68] autour de la cible (Fig. 1.1) ; en effet, lorsqu'un fragment issu de la réaction atteint un détecteur, celui-ci fournit en sortie deux types de signaux électriques transitoires :

- un ou plusieurs signaux logiques indiquant que le détecteur a été touché et permettant de déterminer l'instant auquel il a été atteint.

- un ou plusieurs signaux analogiques dont les caractéristiques sont directement liées à celles du fragment détecté.



F : faisceau .
C : Cible .
Fi : Fragments résultant de la réaction .
Di : Détecteurs .

Fig 1.1 : Représentation schématique de la détection des fragments résultant d'une réaction nucléaire .

Avec ces deux types d'information, on peut donc localiser et caractériser les fragments émis.

1.1.2. Orientation vers les dispositifs multidétecteurs

Les grandes énergies disponibles avec le faisceau du GANIL peuvent donner lieu à des réactions dont la multiplicité, c'est à dire, schématiquement, le nombre de fragments résultants, est élevée (de l'ordre de la dizaine) [DRO 85].

Avec de telles multiplicités, la configuration d'expérience à quelques dizaines de détecteurs utilisée d'ordinaire ne permet plus une caractérisation suffisante des réactions nucléaires mises en jeu, c'est pourquoi, les physiciens ont été amenés à concevoir des ensembles de détection couvrant de grands angles solides. Baptisés "multidétecteurs", ces ensembles sont un assemblage d'un grand nombre de détecteurs individuels d'un même type.

Dans la chambre à réaction "NAUTILUS" seront réunis quatre multidétecteurs : le "Mur de Plastique" [BIZ 86], le "Tonneau", "XYZT" et "DELF" [BOU 87] (fig. 1.2). Le tableau 1.1 donne un bilan du nombre de détecteurs à prendre en compte.

	Mur	Tonneau	XYZT	DELF
Nature des détecteurs	Scintillateur plastique		Plaques Parallèles	
Nombre de détecteurs	96	72	24	18

Tableau 1.1. - Nature et nombre des détecteurs mis en oeuvre sur chacun des multidétecteurs.

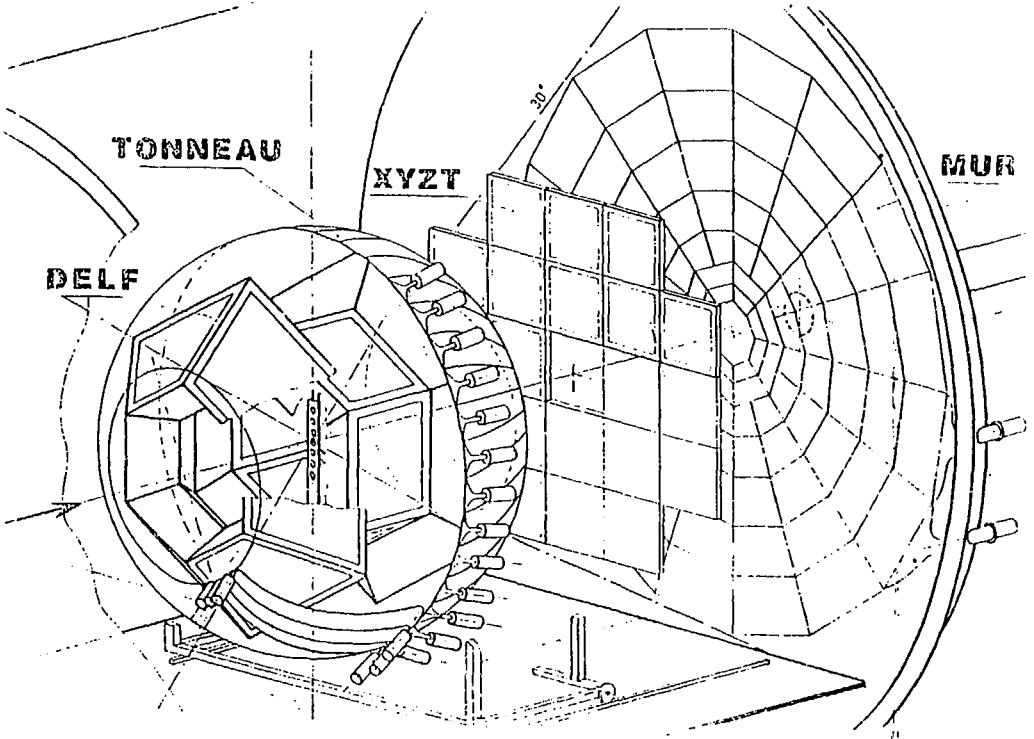


Fig I.2 : Implantation des multidétecteurs dans NAUTILUS

DELF et le TONNEAU détectent les fragments émis dans un angle polaire (par rapport à la direction du faisceau) compris entre 30° et 150° . XYZT et le MUR détectent les fragments émis vers l'avant .

Chaque expérience sera rendue spécifique par adjonction autour du matériel standard que sont les multidétecteurs, de détecteurs fournissant des informations complémentaires sur les réactions produites. On peut d'ores et déjà noter que ces détecteurs, que nous appellerons "additionnels", peuvent se trouver présents en nombre non négligeable : par exemple, l'hodoscope de particules légères dans l'expérience 50 [POC 87] comportait 18 détecteurs (télescopes) fournissant chacun trois signaux.

La somme des informations fournies par chacun des ensembles de détection (multidétecteurs ou additionnel) sera désormais appelée "sous-événement" et la réunion de ceux-ci constituera un "événement".

1.1.3. Acquisition des données et dépouillement

Le système d'acquisition a pour rôle de numériser les signaux fournis par les détecteurs et de sauvegarder les données obtenues, sur bandes magnétiques. Il permet en outre de contrôler en cours d'expérience, le bon déroulement de celle-ci. Une caractéristique essentielle d'un tel système est sa rapidité. En effet, de par le fonctionnement de l'accélérateur, la cible se trouve exposée au faisceau toutes les 100 ns environ. C'est donc à des instants séparés par des intervalles multiples de 100 ns, qu'ont lieu les réactions nucléaires. La distribution de ces intervalles est, par nature, aléatoire. On considère que le système doit, pour être efficace, assurer la prise en compte d'un événement intéressant en un temps inférieur à quelques centaines de microsecondes.

A la fin de l'expérience (qui peut durer plusieurs jours), les physiciens disposent de quelques dizaines de bandes magnétiques sur lesquelles se trouvent toutes les données enregistrées au cours de l'acquisition. Malgré un certain filtrage effectué par le système d'acquisition, une fraction importante de ces données peut ne pas être significative ; la première tâche lors du dépouillement, consiste donc à réduire le nombre de bandes magnétiques en effectuant un filtrage logiciel des événements enregistrés. Dans une deuxième étape, un traitement élaboré des événements retenus permet de corroborer ou d'infirmer les modèles théoriques existants et d'en imaginer de nouveaux.

1.2. Le système d'acquisition actuel

On dispose actuellement au GANIL de quatre ensembles d'acquisition indépendants les uns des autres et construits suivant une même architecture. Ils permettent de réaliser simultanément l'acquisition d'expériences distinctes.

1.2.1. Description générale

Chacun de ces systèmes se compose en allant de l'amont (côté détecteurs) vers l'aval (côté bandes magnétiques), des quatre ensembles fonctionnels suivants (fig. 1.3) :

- Un ensemble de déclenchement appelé configurateur qui, après analyse de la configuration des voies logiques issues des détecteurs, décide d'accepter ou de rejeter l'événement.

- Un ensemble de codeurs chargés en cas d'acceptation de l'événement par le configurateur, de la conversion des signaux analogiques en données numériques. En cas de rejet, le configurateur se remet en attente de l'événement suivant sans déclencher le codage.

- Un microprocesseur (de type CAB : CAMAC Booster) qui organise la lecture des données de l'événement et les place dans une file d'attente (FIFO).

- Un ordinateur (MODCOMP) dont la principale fonction est de lire les événements successifs présents dans la file d'attente et de les stocker sur bandes magnétiques.

Les ensembles de déclenchement, de codage et le microprocesseur sont au standard CAMAC [ESO 83]. L'unité de base de ce standard est le châssis sur lequel on peut enficher différents modules électroniques d'acquisition qui se trouvent alors reliés par un bus de fond de panier appelé "dataway". C'est sur ce bus géré par le "contrôleur de châssis", que transitent les ordres CAMAC à exécuter et les données. Le contrôleur communique avec les différents modules suivant un protocole défini dans la norme. Il fournit et gère notamment les informations suivantes :

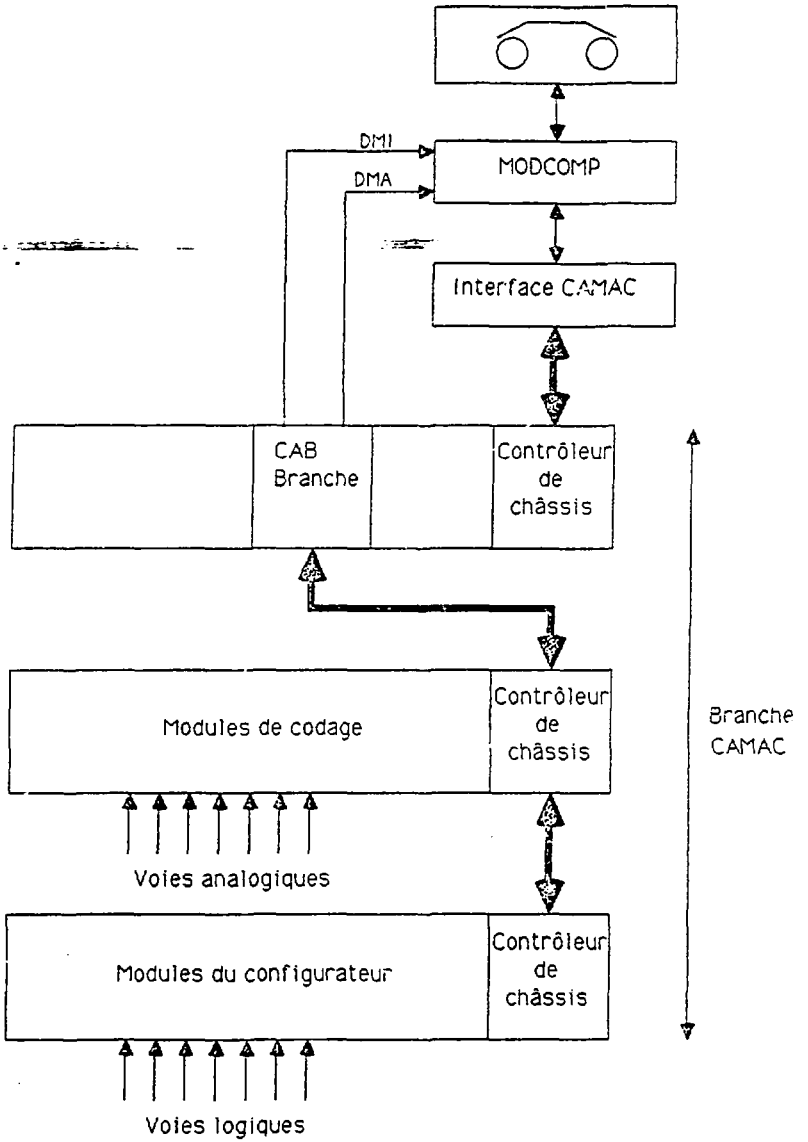


Fig I.3 : Schéma simplifié de l'acquisition GANIL

N : n° de la station où se trouve le module.
A : Adresse à prendre en compte dans le module (registre de données par exemple).
F : Fonction CAMAC à exécuter (lecture, écriture, remise à zéro ...).

Inversement, chaque module peut se signaler au contrôleur en activant une ligne individuelle : LAM (Look At Me). Par ailleurs, le standard prévoit que toute action sur le bus s'exécute en 1 microseconde. De plus, si un seul châssis ne suffit pas pour couvrir tous les besoins en matériel, on peut en placer plusieurs en parallèle sur une interconnexion appelée "branche". Les opérations CAMAC sont alors gérées par un module contrôleur de branche.

Une telle organisation modulaire permet de réaliser facilement des configurations variées d'expérience.

1.2.2. Déclenchement

Une part importante des événements générés lors d'une expérience peut ne pas présenter d'intérêt pour les physiciens et l'acquisition de ces événements représenterait une perte de temps considérable tant à l'acquisition proprement dite qu'au dépouillement. C'est pourquoi on utilise un dispositif de déclenchement qui permet de ne prendre en considération que des événements à priori intéressants.

Le "configurateur", qui joue ce rôle actuellement, est un ensemble de modules fonctionnels qui peuvent être assemblés et programmés de diverses façons en fonction de la sélection demandée par l'expérience [MAR 85]*. Les modes de fonctionnement du configurateur sont multiples, on se bornera donc ici à donner un exemple schématique mais significatif des fonctionnalités qu'il offre (Fig. 1.4).

* Note : les modules du configurateur sont programmés mais, pour des raisons de rapidité, ils fonctionnent en logique câblée.

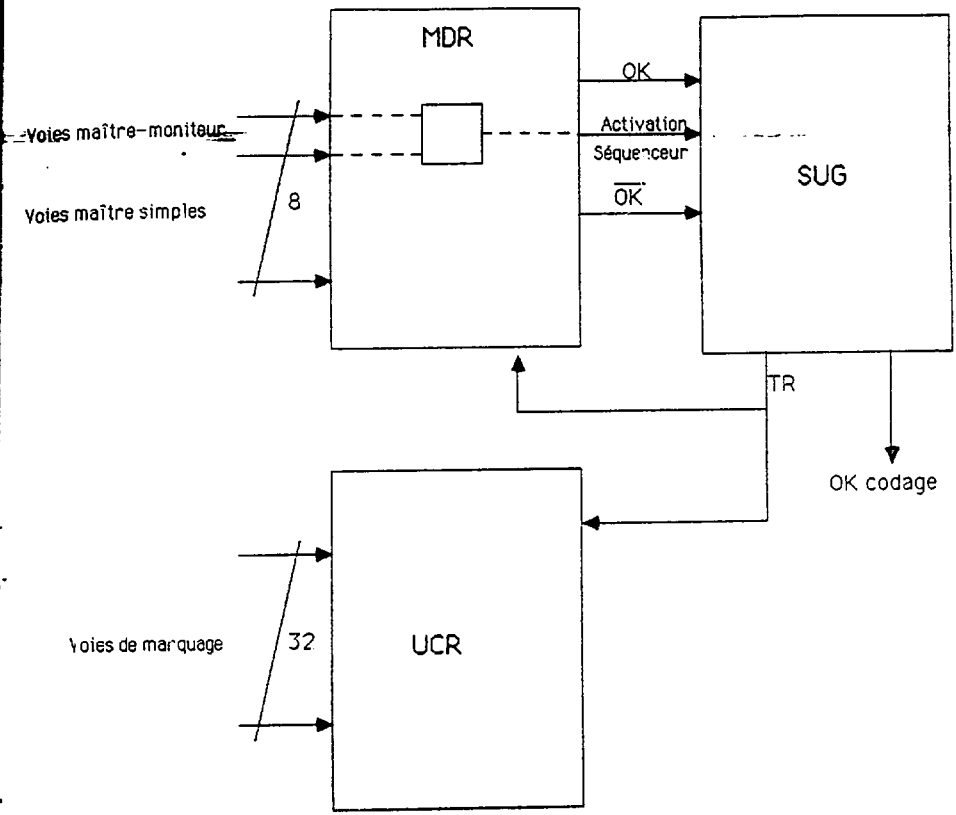


Fig 1.4 : Schéma d'un exemple de déclenchement.

Il a été vu au paragraphe 1.1.1 que les détecteurs placés autour de la cible délivrent un ensemble de signaux analogiques et logiques. Le Module de Décision Rapide (MDR) reçoit parmi ces derniers, les signaux dits "maître-moniteur" qui permettent de lancer le configurateur en activant le Séquenceur d'Usage Général (SUG). Celui-ci, que l'on peut qualifier de chef d'orchestre du déclenchement, fournit alors en retour au MDR, une fenêtre d'analyse TR pendant laquelle il enregistre la configuration de bits présente sur ses huits voies d'entrées. L'analyse de cette configuration permet de décider rapidement (en 150 ns environ) de l'acceptation ou du rejet de l'événement. La décision prise par le MDR est ensuite répercutée sur le SUG. S'il s'agit d'un rejet, le SUG se place en attente de l'événement suivant, sinon, il envoie aux codeurs, un ordre de conversion des signaux analogiques. A ce niveau, deux possibilités se présentent :

1) Les codeurs sont munis de tampons et ces derniers ne sont pas pleins, alors le SUG peut se placer immédiatement en attente de l'événement suivant.

2) Les codeurs ne sont pas tamponnés ou leurs tampons sont pleins, dans ce cas, le SUG doit attendre, que l'événement courant ait été acquis par le CAB pour se remettre en attente. Le temps mort est alors évidemment plus important.

Cet exemple montre les mécanismes mis en jeu pour le déclenchement de l'acquisition mais il faut souligner que certains modules du configurateur n'ont qu'un rôle passif. Ainsi, il peut être intéressant dans certains cas, de mémoriser un certain nombre de voies logiques afin de garder une image de la configuration de l'événement (il s'agit en fait, d'un marquage des détecteurs touchés). Les modules "Universal Coincidence Register" (UCR) permettent de réaliser cette opération : ils mémorisent les signaux pendant une fenêtre de temps et on récupère cette information lors de la lecture par CAMAC de l'événement.

1.2.3. Codage

Les codeurs sont les modules qui assurent la conversion des signaux analogiques délivrés par les détecteurs, en données numériques.

Trois types de grandeurs sont à mesurer : charge, amplitude et temps.

La mesure de charge s'effectue en intégrant le signal sur une fenêtre de temps déclenchée par le Séquenceur d'Usage Général.

La mesure d'amplitude consiste, elle, à relever l'amplitude maximum atteinte par le signal pendant une fenêtre.

Le temps se mesure par un codage approprié de la durée qui sépare un signal de départ et un signal d'arrêt, l'un des deux étant en général fourni par le SUG.

Ces modules sont lus par le microprocesseur CAB via la branche CAMAC.

1.2.4. Le microprocesseur CAB

Dans un premier temps, la seule unité centrale du système d'acquisition était l'ordinateur MODCOMP qui assumait à la fois les tâches d'acquisition, de sauvegarde des données sur bande et de contrôle en ligne de l'expérience. Il s'est avéré que l'on pouvait améliorer de façon considérable les performances du système en implantant dans la branche CAMAC un microprocesseur spécialisé en acquisition.

Le microprocesseur retenu pour cette application : le CAB est un microprocesseur en tranches développé spécialement pour effectuer des acquisitions de données par CAMAC, à grande vitesse [MAT 83]. Une instruction CAB s'exécute en environ 250 ns. Placé en position de contrôleur sur la branche CAMAC, il exécute les logiciels d'acquisition chargés dans sa mémoire par le calculateur MODCOMP à l'initialisation. Ces derniers deviennent alors complètement autonomes vis à vis de l'ordinateur.

D'autre part, le CAB est capable d'exécuter quatre instructions pendant le déroulement d'un cycle CAMAC (1 µs). On peut donc profiter de cette possibilité pour effectuer un premier traitement des données en cours d'acquisition. Suivant la configuration logique recueillie par exemple, on peut décider de lire tel ensemble de codeurs

plutôt qu'un autre ce qui évite des transferts inutiles [GUE 84]. On peut aussi décider du rejet d'un événement sur analyse de configuration ou par comparaison des données lues, à des fenêtres ou des contours. Le dernier exemple que l'on donnera, est la possibilité, sur les mêmes critères que précédemment, de pratiquer un échantillonnage des événements transmis à l'ordinateur.

On obtient donc après insertion du CAB, un dispositif d'acquisition à la fois plus rapide et plus souple.

Pour optimiser encore l'utilisation du CAB, on lui adjoint une mémoire tampon dans laquelle il peut stocker les événements en file (en attendant leur lecture par le MODCOMP) et deux interfaces de communication avec ce dernier [TIL 83]. Il serait en effet trop long de transmettre les données au MODCOMP à travers l'interface CAMAC, c'est pourquoi ont été étudiées une liaison DMA et une liaison DMI.

La liaison DMA permet un transfert rapide de paquets d'événements vers les mémoires du MODCOMP, la liaison DMI quant à elle, permet d'effectuer une incrémentation automatique et rapide d'histogrammes. Un histogramme est un spectre de fréquence pour une variable analysée et le module DMI procède de la façon suivante : la donnée reçue est considérée comme une adresse dont on incrémente automatiquement, en logique câblée, le contenu d'un.

Toutes ces améliorations ont permis de doubler les performances en vitesse du système, le taux d'acquisition passant de 30 000 à 60 000 mots par seconde.

1.2.5. Le rôle de l'ordinateur

C'est à travers un ordinateur MODCOMP 16 bits que s'opère tout le contrôle de l'expérience en cours grâce à un moniteur d'acquisition appelé GAMAG [VAI 83].

Il effectue en premier lieu, les opérations nécessaires au démarrage de l'acquisition, notamment : initialisation des modules CAMAC, chargement des logiciels d'acquisition dans le CAB, chargement et lancement des logiciels à exécuter.

En fonctionnement normal de l'acquisition, trois tâches (au sens système du terme) sont activées. La première, que l'on appelle tâche de lecture, stocke les événements reçus par DMA dans des files d'attente. Un mécanisme de gestion producteur/consommateur permet alors à une deuxième tâche, dite "d'écriture", de lire les données enregistrées et de les sauvegarder sur bande. La troisième tâche effectue des traitements sur les événements et permet de construire et de visualiser des spectres. Ces derniers permettent de contrôler le bon déroulement de l'expérience ; par exemple, un canal qui ne s'incrémente plus, peut signifier un codeur défectueux qu'il faut remplacer.

En fin d'expérience, le calculateur assure l'arrêt de l'acquisition.

1.3. Définition d'un cahier des charges pour le système d'acquisition des multidétecteurs

1.3.1. Caractéristiques générales du système

Compte tenu du nombre important de voies à lire pour chaque événement (on compte sur une centaine de paramètres par événement), il paraît primordial que le système soit rapide. Le temps mort est un élément caractéristique de la vitesse d'un système d'acquisition, qui indique la durée moyenne pendant laquelle ce dernier se trouve bloqué pour le transfert et la sauvegarde d'un événement. En rapportant ce temps au nombre de voies lues, on obtient le temps mort par paramètre qui, dans notre application, ne devra pas dépasser 4 µs. En outre, le débit maximum admis par les dérouleurs de bandes étant de 150 k mots/s, il sera nécessaire de procéder à un filtrage des événements en ligne afin d'éliminer le plus tôt possible dans la chaîne d'acquisition, les données sans intérêt. Le système devra donc prendre en compte un maximum d'événements en entrée et ne fournir en sortie que les plus intéressants.

Dans cet esprit, on munira le système d'un dispositif de déclenchement effectuant un premier filtrage des événements

(cf. § 1.2.2.). Ce dispositif devra être rapide et pouvoir s'adapter à toutes les combinaisons possibles d'utilisation des multidétecteurs.

Ce dernier critère reste vrai pour l'ensemble du système d'acquisition dont les performances et les fonctionnalités devront de plus, pouvoir s'adapter en permanence à l'évolution de la physique pratiquée avec les multidétecteurs.

Les matériels et les logiciels mis en oeuvre dans les expériences "multidétecteurs" seront complexes et leur mise au point pourra conduire à des préparatifs d'expérience relativement longs. Afin d'éviter de monopoliser un des ensembles d'acquisition pour ces tests, on prévoit un dispositif de contrôle et de visualisation intégré au système et indépendant du MODCOMP. Ce dispositif sera aussi utilisé en expérience et devra alors être suffisamment rapide pour ne pas ralentir l'acquisition. Dans un tel contexte, l'ordinateur sert juste "d'interface" avec les dérouleurs de bandes dont il est muni. Il suffira donc d'adjoindre des outils de stockage (dérouleur de bandes, disque optique) au système d'acquisition des multidétecteurs pour faire de celui-ci un ensemble parfaitement autonome.

Pour être vraiment efficace, le système devra être convivial et permettre à l'utilisateur d'exploiter par des commandes simples toutes les fonctionnalités qui lui seront proposées. D'autre part, le système supportera différents langages évolués connus des utilisateurs (FORTRAN, C, par exemple) et ces derniers disposeront de points d'entrée dans les logiciels où ils pourront intégrer leurs propres programmes d'acquisition ou de traitement.

Dans la réalisation du système, on s'attachera à utiliser un minimum de standards différents et un maximum de matériels et de logiciels existant sur le marché ; ceci afin d'éviter des développements souvent longs et de faciliter la maintenance des dispositifs ainsi que le suivi des techniques nouvelles.

On terminera ces caractéristiques générales par un point "d'intendance" non négligeable. La prise en compte des quelques 700

paramètres des multidétecteurs représentera un volume de matériel imposant qu'il semble peu rationnel pour des questions de temps et de fiabilité de monter et de démonter à chaque nouvelle expérience. Les multidétecteurs et les modules d'acquisition qui leur seront liés resteront donc implantés à poste fixe et seules les voies additionnelles seront montées et démontées à chaque expérience.

1.3.2. Caractéristiques de codage

De la même façon qu'actuellement, on mesure des charges, des temps et des amplitudes ; le tableau 1.2 donne à titre indicatif la répartition de ces trois types de voies sur les multidétecteurs. Afin de

	Mur	Tonneau	XYZT	DELF
Q	96	144	24	18
T	96	144	120	90
E	0	0	24	18

Tableau 1.2 : Nombre de voies à prendre en compte en fonction de leurs types pour chaque multidétecteur.

limiter les temps morts, les codeurs devront permettre une lecture rapide et sélective car on ne veut pas lire les voies non touchées. Leurs performances analogiques devront s'approcher de celles indiquées dans le tableau 1.3.

	Charge	Temps	Amplitude
Codage	11 bits	11 bits	12 bits
Linéarité différentielle	± 10 %	± 10 %	± 1 %
Pleine échelle	250 pc	500 ns	10 V
Temps codage	10 µs	10 µs	10 µs
RAZ	200 à 300 ns	200 à 300 ns	200 à 300 ns

Tableau 1.3 : Cahier des charges pour les caractéristiques analogiques des codeurs.

Signalons pour terminer, que compte tenu du parc de codeurs à créer, les prix de ces derniers seront un argument important.

CHAPITRE 2

LES BASES DU SYSTEME D'ACQUISITION DES MULTIDETECTEURS

Après avoir défini le cahier des charges du système d'acquisition des multidétecteurs, nous devons trouver un dispositif qui s'y conforme le mieux possible. L'expérience montre que le système d'acquisition actuel n'est, malheureusement, pas adapté à notre application et qu'il se révèle nécessaire d'axer les développements sur une philosophie différente dont nous donnerons les grands traits. Nous pourrons alors, pour terminer, entrer dans le vif du sujet, avec la description du front d'acquisition composé des modules de déclenchement et de codage.

2.1. Le système d'acquisition actuel est-il adapté aux multidétecteurs ?

Des essais effectués avec le Mur de plastique montrent que ce n'est pas le cas. Le système se révèle en effet, manquer de rapidité pour réaliser des acquisitions de type multidétecteurs. Dans l'expérience 50 par exemple [POC 87], 250 paramètres étaient à prendre en compte et, avec une longueur moyenne d'événement de l'ordre de la cinquantaine de mots, le temps mort par paramètre atteignait 12 µs ce qui dépasse largement les bornes imposées par le cahier des charges.

On peut observer par ailleurs, des limitations dans l'utilisation du système, en effet, le CAB est un outil rapide mais difficile d'accès car peu convivial. De plus, sa mémoire d'instructions étant limitée à 4 K instructions, on peut se trouver limité lors du chargement d'un ou plusieurs logiciels conséquents.

On devra donc étudier un dispositif mieux adapté tant en rapidité qu'en souplesse d'utilisation.

2.2. Les principes généraux de conception du système d'acquisition

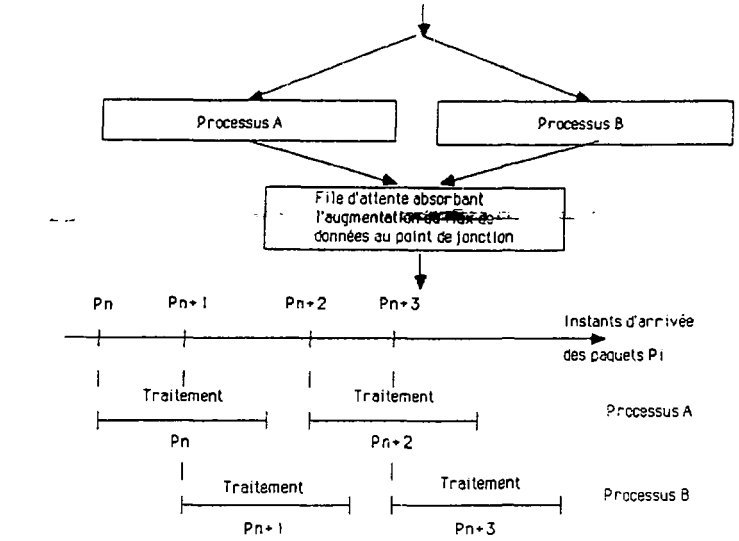
Les principes exposés ici s'appliquent particulièrement bien pour la réalisation de tout système effectuant l'acquisition d'un nombre important de paramètres, tant en physique nucléaire qu'en physique des hautes énergies [GAV 85, LEC 85].

2.2.1. Parallélisme et asynchronisme

Considérons un système dans lequel le flux d'information se présente sous forme de paquets de données successifs.

Dans une architecture parallèle (Fig. 2.1.a), deux ou plusieurs unités fonctionnelles identiques, situées à un même niveau du système effectuent simultanément et indépendamment les unes des autres, un même traitement sur des paquets d'information distincts. Une telle organisation requiert un algorithme de gestion permettant de déterminer pour chaque nouveau paquet incident, quelle unité doit le traiter. Si le temps d'exécution de cet algorithme reste négligeable devant celui du traitement des données, le gain en vitesse peut se révéler important car tout se passe alors comme si le temps de traitement de chaque paquet d'information se trouvait divisé par le nombre d'unités mises en oeuvre. Ce dernier n'est donc limité que par deux choses : d'une part, la vitesse d'exécution de l'algorithme de gestion (qui peut diminuer lorsque le nombre n d'unités utilisées croît) et d'autre part, le prix de l'unité, qui multiplié par n peut devenir prohibitif.

On peut par ailleurs envisager des situations où, sans qu'aucun parallélisme ne soit appliqué, plusieurs unités fonctionnelles opèrent simultanément. Considérons comme indiqué figure 2.1.b, deux processus A et B fonctionnant l'un en amont du tampon T et l'autre en aval. Le processus A reçoit le paquet d'information n, le traite et le place dans le tampon. Dès la fin de cette opération, il peut se consacrer au paquet n + 1 mais le processus B peut alors dans le même temps lire puis traiter les données du paquet n contenues dans le tampon. Les processus A et B qui généralement, effectuent des actions distinctes, opèrent donc séquentiellement pour un paquet donné mais de façon totalement asynchrone l'un par rapport à l'autre puisque B traite



Pi = Paquet de données n° i.

Fig 2.1.a : Architecture parallèle et chronogramme associé

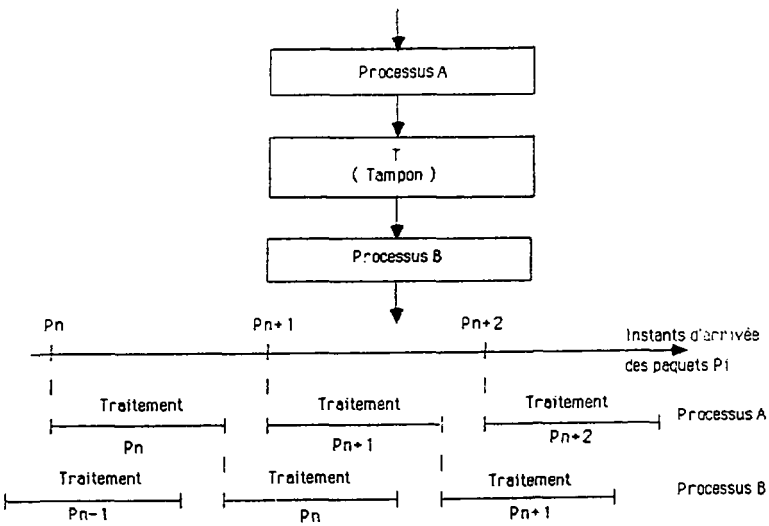


Fig 2.1.b : Organisation producteur-consommateur asynchrone et chronogramme associé

le paquet n alors que A prend déjà le paquet n + 1 en compte. Il existe donc entre eux une relation producteur/consommateur que l'on ne retrouve pas dans une organisation parallèle. Dans le système d'acquisition actuel, l'accroissement des performances résulte de la mise en place d'une architecture producteur/consommateur asynchrone avec le microprocesseur CAB dédié à l'acquisition de données placé en frontal du calculateur MODCOMP qui sauvegarde les événements sur bandes.

Dans le système d'acquisition des multidétecteurs, on placera les unités de filtrage notamment, sur des branches parallèles, et entre tous les étages du dispositif, on installera des files d'attente ce qui permettra aux unités aval de fonctionner indépendamment des unités amont.

2.2.2. Utilisation de files d'attente

La fonction première d'une file d'attente est de lisser le flux des données dans le temps. En effet, si à un instant donné, on a un afflux brutal d'événements que les dispositifs placés en aval ne sont pas capables d'absorber, ce surcroît d'informations doit être stocké dans des mémoires en attendant d'être lu [VOL a 80, VOL b 80].

On peut ainsi, en particulier, améliorer l'efficacité d'une organisation asynchrone, en intercalant une file d'attente plutôt qu'un tampon unique entre deux processus. Le processus amont charge alors les informations à la queue de la file tandis que le processus aval vient lire les données présentes en tête de file. Les seuls problèmes de synchronisation qui se posent alors interviennent lorsque la file est vide ou pleine, auquel cas, il faut interrompre respectivement l'unité aval ou l'unité amont.

De la même façon, au point de jonction de plusieurs branches parallèles, les flux de données des différentes branches s'ajoutent et on peut avoir un phénomène "d'étranglement" avec perte d'informations si l'on ne prévoit pas de mémoire tampon (Fig. 2.1.a).

Ces deux types de problème se trouvent présents dans notre application puisque, d'une part, les événements se répartissent aléatoirement dans le temps (ceci étant lié à la physique) et que d'autre part, en un point du système, on devra rassembler les données fournies par les différents multidétecteurs.

2.2.3. Filtrage en ligne

La nécessité d'un filtrage des données en cours d'acquisition a déjà été mise en évidence dans le cahier des charges du système, le but de ce paragraphe est de préciser certains aspects de ce filtrage.

On peut noter tout d'abord que si l'on n'effectuait aucune sélection des événements enregistrés, on passerait beaucoup de temps à transférer des données inutiles ce qui ralentirait l'acquisition et compliquerait le dépouillement. Si par contre, chaque étage du système d'acquisition pratique un filtrage des événements sans intérêt, le flux de données diminue lorsqu'on va de l'amont vers l'aval du système. Chaque étage passe par conséquent, moins de temps à effectuer des transferts que l'étage précédent et peut donc opérer sur les données, des traitements plus élaborés (Fig. 2.2). Par conséquent, tant que le flux de données reste important, on se limite à des filtrages rapides, basés sur des critères simples et réalisés en logique câblée. Mais dès que possible, on doit introduire des filtrages logiciels qui, bien que moins rapides sont plus souples d'emploi, plus évolutifs et plus sélectifs.

Il reste cependant important de se réserver la possibilité d'inhiber certains processus de filtrage pour des contrôles ou dans des conditions particulières (comme une recherche de panne par exemple).

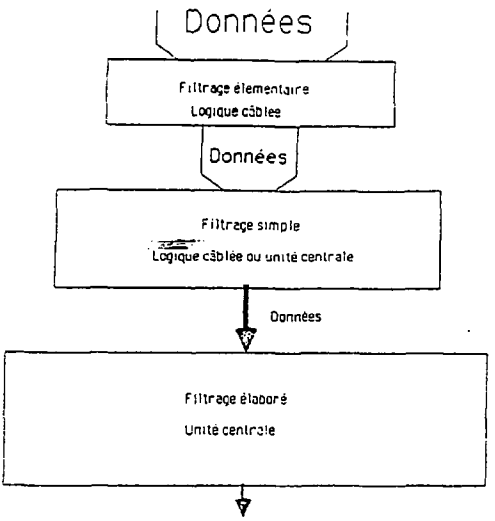


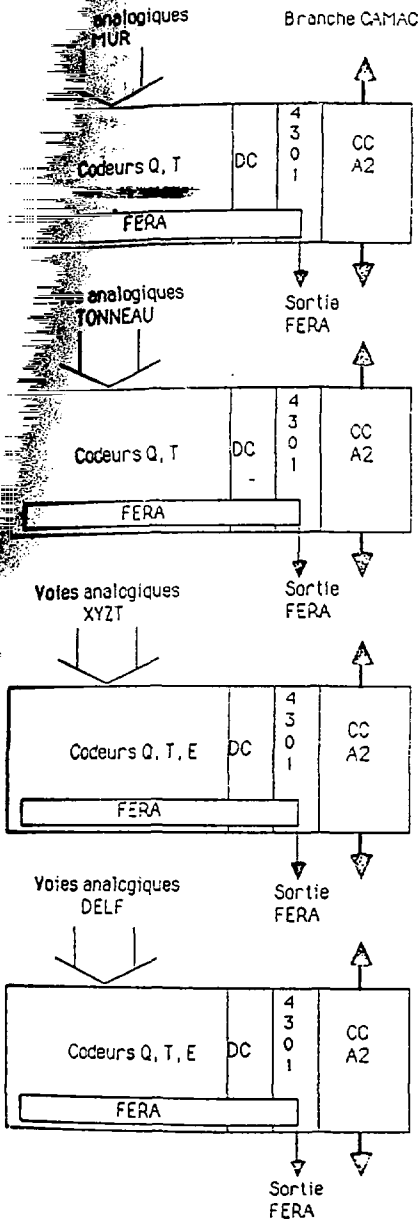
Fig 2.2 : Evolution du flux de données et du type de filtrage effectué en allant de l'amont vers l'aval du système

2.2. L'organisation des modules CAMAC

2.3.1. Présentation générale

Pour la réalisation du front de l'acquisition (déclenchement et codage), le standard CAMAC a été retenu car il est bien connu des utilisateurs et offre un large choix de matériels parmi lesquels on peut trouver ceux qui répondent à nos besoins.

La figure 2.3 donne le schéma général de l'organisation des modules CAMAC. Le configurateur, rapide, souple d'emploi, modulaire et adaptable à des types très variés d'expérience répond bien, à l'heure actuelle, au cahier des charges défini pour le dispositif de déclenchement de l'acquisition et sera donc utilisé comme tel dans le système. D'autre part, afin qu'ils restent indépendants les uns des autres, les multidétecteurs disposent chacun de leur propre ensemble de codage et pour des questions de rapidité, ce dernier est lu non par



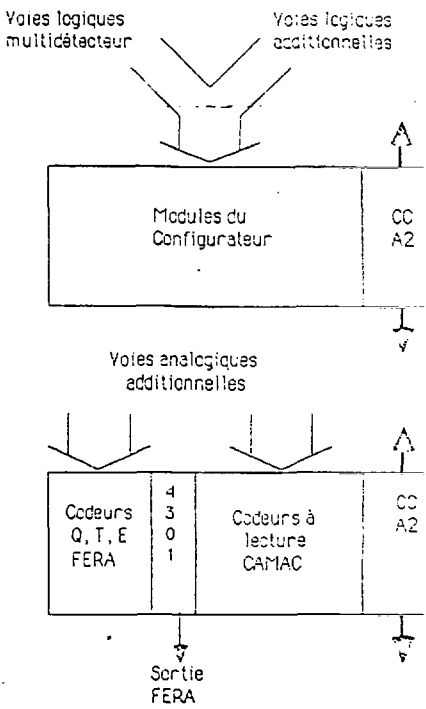


Fig 2.3 : Schéma général de l'organisation des modules CAMAC.

CAMAC mais par un bus en face avant : le bus FERA (Fast Encoding and Readout ADC). Les cartes DC (Décisions Câblées) espionnent ce bus et peuvent décider, en fonction des valeurs prises par les données qui y circulent, du rejet de l'événement. On peut, par ailleurs, déjà noter qu'une partie des voies (configurateur, voies additionnelles) devront être lues par CAMAC ce qui posera des problèmes.

Signalons pour terminer, que le nombre de modules à mettre en place est tel que l'on devra les répartir sur deux branches CAMAC (c'est-à-dire au moins 8 châssis).

2.3.2. Les codeurs

Une étude des possibilités de codage a été réalisée et a permis de dégager trois axes différents [RIC 85].

Un codeur de temps, tout d'abord ; développé par l'Institut des Sciences Nucléaires de Grenoble, ce codeur offre des caractéristiques analogiques correctes mais, ne pouvant être lu que par CAMAC, il n'a pas été retenu. Les codeurs SILENA pour leur part présentent de très bonnes performances analogiques et disposent d'un bus rapide de lecture en face avant. Ils sont malheureusement environ quatre fois plus chers que les codeurs LECROY de la série 4300 [LECR] qui ont finalement été retenus pour les mesures de charge et de temps malgré les écarts entre leurs caractéristiques analogiques (données tableau 2.1) et le cahier des charges.

Des essais effectués sur un ensemble 4300 (Annexe 1) donnent les résultats suivants pour un codeur de charge 4300 B (11 bits) :

- linéarité différentielle : ± 15 %
- le temps de codage est de 8,5 µs mais les données ne sont disponibles sur le bus que 1 µs plus tard ce qui n'est pas négligeable.

	Charge (4300)	Temps (4303)
Codage	10 ou 11 bits	
Linéarité différentielle	Typ : ± 10 % Max : ± 20 %	
Pleine échelle	480 pc (11 bits)	1024 ns
Temps de codage	8,5 µs	
Temps de RAZ	2 µs	

TABLEAU 2.1
Caractéristiques analogiques des codeurs de la série 4300

L'intérêt majeur de ces codeurs est en fait la présence d'un bus de lecture rapide en face avant. Ce bus, réalisé en technologie ECL permet une lecture des données à raison de 100 ns par mot et fonctionne de la façon suivante (Fig. 2.4). Les codeurs sont reliés à un module contrôleur (4301) par un bus de données et un bus de contrôle qui permet de gérer les ordres de codage, la lecture et la remise à zéro des modules. Un cycle FERA commence par le signal "GATE", distribué par le 4301 aux codeurs, qui sert à la fois d'ordre de début de codage et de fenêtre de temps pour la prise en compte des signaux analogiques. En fin de numérisation, les codeurs signalent au 4301 que les données sont disponibles en positionnant la ligne "REQ" (Request). Ce dernier fournit alors un signal "REN" (Readout Enable) qui, circulant le long d'une "daisy-chain", autorise les codeurs les uns après les autres à délivrer leurs données sur le bus. Lorsque tous les codeurs ont été lus, le contrôleur positionne la ligne "CLR" (Clear) du bus de contrôle et les codeurs sont alors remis à zéro.

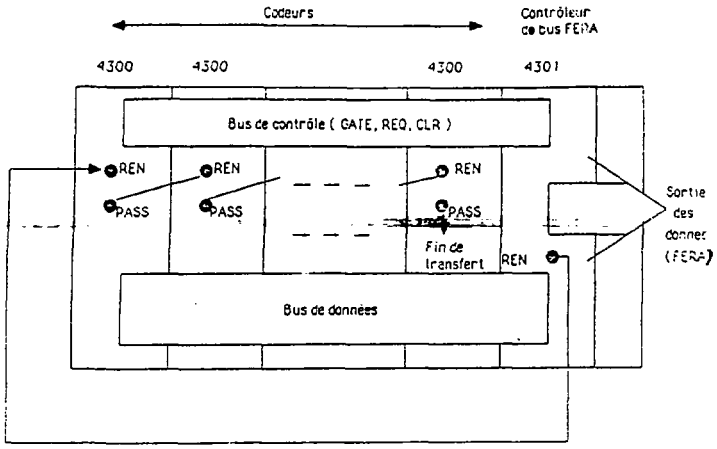


Fig 2.4 : Schema de câblage et de fonctionnement du bus FERA

VOIE 0	0	Donnée
VOIE 1	0	Donnée
VOIE 15	0	Donnée

Format des données sans suppression des zéros

ENTETE	1	WC	COD	VSN
1ère VOIE	0	SA	Donnée	
DERNIERE VOIE	0	SA	Donnée	

VSN : Virtual Station Number (donne la provenance des données) .
WC : Word Count (nombre de voies transmises) .
COD : Codage de la résolution (10, 11, 12 ... bits) .
SA : Sub-Address (sous-adresse) .

Format des données avec suppression des zéros

Fig 2.5 : Format de sortie des données au standard FERA

Le format des données sur le bus FERA est indiqué figure 2.5. On voit que l'on peut, soit lire systématiquement toutes les voies d'un codeur, soit éliminer dès la lecture, toutes les voies dont la valeur se situe au-dessous d'un seuil programmé. Cette suppression de zéros fait passer le temps de codage du 4300 de 9,3 à 11,5 µs mais reste intéressante comme le montre l'exemple suivant :

Considérons le TONNEAU qui comporte 72 détecteurs représentant 144 voies de codage en charge et 144 voies en temps. Si toutes les voies sont lues, on a alors le temps (codage + lecture) suivant :

$$9,3 + (144 \times 0,1) \times 2 = 38,1 \ \mu s$$

Si on considère que sur les 288 voies mises en jeu, 50 seulement (mais c'est déjà beaucoup) sont valides, le codage et la lecture de ces voies prend :

$$11,5 + 50 \times 0,1 = 16,6 \ \mu s$$

soit deux fois moins que précédemment.

Dans le format de sortie avec suppression de zéros, on dispose de plus, d'informations complémentaires sur les données. Un mot d'entête repéré par le bit 16 à 1 indique, dans son octet de poids faible, la provenance des données ("Virtual Station Number") qui est en fait un numéro attribué au codeur à l'initialisation. Dans les trois bits suivants qui ne sont pas utilisés par LECROY, on prévoit de coder la précision des données : 11, 12, 13 ... bits, enfin on donne dans les quatre bits restants, le nombre de données transmises ("Word Count"). Pour un codeur, le numéro de la voie d'où est issue chaque donnée est fourni dans les bits de poids fort (le bit 16 restant à 0 pour les données). Cette indication est appelée Sous-Adresse (SA) et le nombre de bits utilisables pour la coder dépend de la précision des données.

En ce qui concerne les mesures d'amplitude, il a été développé au GANIL en collaboration avec l'ISN de Grenoble un codeur répondant aux spécifications analogiques demandées et compatible avec le bus FERA [TRI 87], ce module est actuellement opérationnel et sera implanté dans le système.

Les codeurs seront, pour chaque multidétecteur, regroupés par type dans des châssis CAMAC (Fig. 2.6).

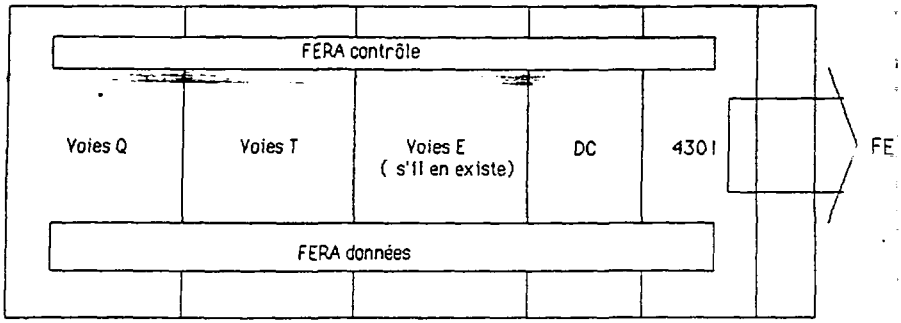


Fig 2.6 : Organisation des modules de codage pour un multidétecteurs

2.3.3. Les cartes DC

La conception des cartes DC s'inspire de celle des cartes "X" implantées dans le système d'acquisition du multidétecteur "Château de Cristal" à Strasbourg [VIL 84]. Il s'agit de placer sur le bus FERA, un premier niveau de filtrage des événements juste après codage. Le nombre de données mises en jeu étant encore très important, ce filtrage sera, pour rester suffisamment rapide, réalisé en logique câblée. Chacun des multidétecteurs dispose d'une carte DC propre, pouvant fournir un avis d'acceptation ou de rejet sur les sous-événements qui circulent sur le bus FERA. Les décisions prises par ces différentes cartes sont ensuite envoyées sous forme de signaux logiques, vers un module réalisant des fonctions logiques simples, qui décide alors de l'acceptation ou du rejet de l'événement complet.

Des discussions avec les futurs utilisateurs ont permis de définir les fonctionnalités précises des modules DC, ce qui a conduit à l'étude de trois cartes différentes : la première pour le MUR, la seconde pour le TONNEAU et la troisième pour XYZT et DELF (qui utilisent des détecteurs d'un même type). Bien que différenciées dans leur réalisation, ces cartes ont un principe de commun que nous allons décrire ici.

Une première caractéristique importante des cartes DC est qu'elles ne ralentissent pas le transfert d'informations sur le bus FERA. Placées en position d'espion sur cette interconnexion, elles prennent les données au vol et leur électronique très rapide (réalisée en technologie ECL) leur permet de traiter une donnée avant l'arrivée de la suivante sans introduire de temps mort supplémentaire.

La carte DC vérifie essentiellement, pour chaque détecteur touché, que les différentes données qu'il fournit, ont des valeurs qui s'intègrent bien dans un ensemble de fenêtres. Celles-ci sont définies à l'initialisation et chargées dans une mémoire du module. La figure 2.7 présente l'exemple de la carte DC associée au MUR. Le résultat des comparaisons effectuées permet d'une part, de ne prendre en compte, pour le calcul de la multiplicité, que des détecteurs présentant un intérêt physique et d'autre part, de décider d'accepter ou de rejeter le sous-événement courant. Cette dernière fonction constitue l'essentiel du mode de fonctionnement de la carte DC. Celle-ci peut à nouveau, après transfert sur le bus FERA de toutes les données issues des codeurs, émettre un avis d'acceptation ou de rejet du sous-événement, par comparaison de la multiplicité obtenue avec une valeur fixée à l'initialisation (Fig. 2.8).

Il faut préciser ici qu'une carte DC, de par sa "transparence" vis à vis des données circulant sur le bus FERA, n'a pas le pouvoir de rejeter effectivement un sous-événement, c'est-à-dire de bloquer d'une façon ou d'une autre les données qui constituent celui-ci. Le signal traduisant la décision d'un module DC permet à un module logique de centralisation (Fig. 2.9) de ces décisions, d'accepter ou de rejeter l'événement complet et ce dernier est alors soit intégralement transmis, soit intégralement bloqué dès son arrivée dans les tampons d'entrée du système d'acquisition.

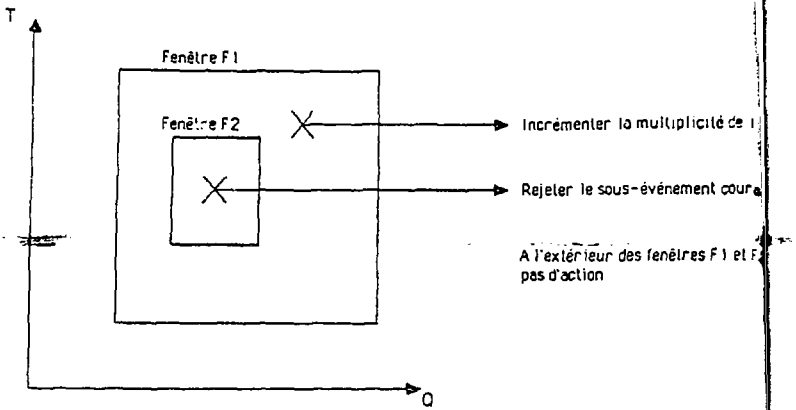


Fig 2.7 : Décision prise par la carte DC associée au MUR en fonction de la valeur prise par le couple (Q, T) issu de chacun des détecteurs touchés

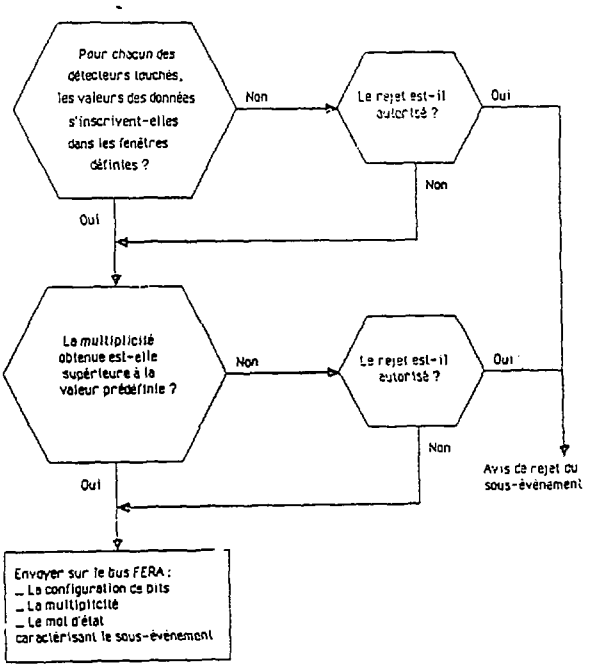


Fig 2.8 : Organigramme de fonctionnement de la carte DC associée au MUR de PLASTIQUE

Cependant la carte DC n'est pas seulement un organe de filtrage, elle effectue, également, en ligne, des traitements élémentaires dont les résultats, émis sur le bus FERA en cas d'acceptation du sous-événement, viennent s'ajouter à ce dernier. Elle délivre notamment deux informations intéressantes pour les traitements en aval du système. Il s'agit d'une part d'une configuration de bits indiquant le marquage des détecteurs touchés et d'autre part de la multiplicité.

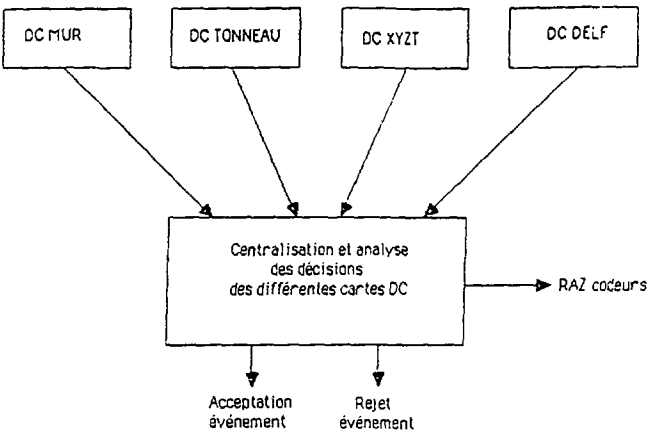


Fig 2.9 : Centralisation des décisions prises par les cartes DC

On dispose sur chaque carte, d'une option qui inhibe la sortie des signaux de rejet, ce qui permet d'enregistrer des événements qui normalement auraient dûs être rejetés. Un mot d'état fourni par la carte en même temps que la configuration de bits et la multiplicité, indiquant pourquoi ces événements devraient être rejetés (fenêtre, multiplicité), permet de les distinguer des événements effectivement acceptés et peut être utilisé pour des traitements logiciels.

Ce principe de fonctionnement est très ouvert puisque rien n'empêche, sur le plan du principe, de placer sur un même bus FERA, plusieurs cartes OC assumant des tâches distinctes. Toutefois, il faut noter que ces modules occuperont chacun trois emplacements CAMAC et que ceci peut représenter une limitation pratique à l'utilisation de plusieurs d'entre eux sur un même bus.

2.3.4. Les problèmes liés à la lecture des voies additionnelles

Installées dans NAUTILUS avec les multidétecteurs, ces voies fournissent des informations complémentaires sur les réactions mises en jeu. Les modules électroniques d'acquisition qui leur seront associés dans un châssis CAMAC à part, ne disposeront pas forcément d'un bus de lecture FERA et il faudra les lire par l'intermédiaire de la branche CAMAC. Ceci pose en premier lieu un problème de rapidité car la lecture de ces voies (1 µs par lecture CAMAC) sera au moins dix fois plus lente que celle des données issues des multidétecteurs (100 ns par lecture FERA). L'acquisition se trouvera donc d'autant plus ralentie que le nombre de voies à lire par CAMAC sera plus important. Par ailleurs, le format des données lues par CAMAC sera totalement différent de celui des données lues par FERA et ce manque d'homogénéité pourra compliquer les programmes de traitement des événements.

Il paraît à l'heure actuelle, impossible de s'affranchir totalement des lectures par CAMAC, on cherchera donc, pour limiter le temps mort, à utiliser partout où cela sera possible, des modules au standard FERA pour le codage des voies additionnelles.

Compte tenu du fait que ces voies pourront avoir un rôle très différent d'une expérience à l'autre, il ne paraît pas, pour l'instant, possible de définir les fonctionnalités d'une carte OC qui interviendrait sur le bus FERA "additionnel".

CHAPITRE 3

DETERMINATION D'UNE ARCHITECTURE MATERIELLE

Il s'agit dans ce chapitre, de définir un dispositif capable de lire et de traiter de façon élaborée, les informations disponibles sur les différents bus FERA ainsi que sur la branche CAMAC. En la matière, comme en témoigne le nombre de systèmes d'acquisition différents existant en physique nucléaire, le champ des possibilités est très étendu. Cependant, on constate de plus en plus, dans ces systèmes, une évolution vers une décentralisation de "l'intelligence" sur l'ensemble du dispositif. Ainsi, dans un souci de rapidité, différentes unités centrales réparties en des points bien choisis du système, effectuent simultanément des travaux spécifiques. Les microprocesseurs, par leur modularité, leur souplesse d'emploi et leur faible coût s'intègrent parfaitement dans ce type d'organisation, c'est pourquoi nous articulerons notre architecture autour de ce type de composants. Nous verrons par ailleurs, que l'on peut obtenir des performances très différentes suivant que le regroupement des sous-événements s'effectue plutôt en amont ou plutôt en aval du système d'acquisition. Nous décrirons en particulier, un modèle d'organisation dans lequel chaque multidétecteur se trouve pourvu d'un ensemble de filtrage logiciel et nous indiquerons les raisons qui nous ont orientés vers une structuration mieux adaptée du matériel, dans laquelle les sous-événements sont regroupés avant tout traitement logiciel.

3.1. Choix des microprocesseurs

3.1.1. La famille 68000

Afin de profiter au maximum des technologies les plus récentes, c'est avec un esprit totalement nouveau, par rapport à la conception du 6800, que la firme MOTOROLA a lancé l'étude et la réalisation du 68000. Il en est ressorti un microprocesseur puissant muni d'une dizaine de modes d'adressage différents qui permettent

une grande souplesse de traitement et de transfert des données ; ceci d'autant plus que toutes les adresses font office d'accumulateur au sens 6800 du terme ce qui signifie que l'on peut réaliser des opérations sur leur contenu sans passer par l'intermédiaire d'un registre spécial.

Depuis la sortie du premier 68000, MOTOROLA a produit de nouvelles unités centrales plus performantes ou plus spécifiques. Ainsi, du 68000 au 68020 en passant par le 68010 pour ne citer qu'eux, on dispose d'une palette de composants adaptés à des besoins qui peuvent être différents.

Le 68000, produit de base, est un microprocesseur 16 bits rapide et souple avec un jeu d'instructions performant qui en fait un bon outil tous "usages".

Le 68010 dispose de registres de contrôle supplémentaires et surtout d'un système de "pipe-line" qui lui permet de stocker dans une mémoire interne, le code de trois instructions simples. Ceci est particulièrement intéressant pour l'exécution de boucles de transfert de données car le microprocesseur vient chercher une fois dans la mémoire externe, le code de la boucle et celle-ci est ensuite exécutée de façon purement interne avec le gain de temps que cela implique. L'emploi de processeurs 68010 s'applique donc bien à des systèmes à vocation de transferts de données.

Tout en restant dans la même ligne que ses prédécesseurs, le 68020 qui est un microprocesseur 32 bits, offre encore plus de registres de contrôle, plus de modes d'adressage et plus d'instructions. Il possède en outre, une mémoire dite "cache" qui lui permet de conserver en interne 128 mots de 16 bits d'instruction. Sachant qu'une instruction peut comprendre de un à onze mots, on voit que le code contenu dans la mémoire cache peut être assez important et que l'on peut donc, en particulier dans l'exécution de boucles, gagner beaucoup de temps. Ce type de microprocesseur reste tout à fait performant en vitesse puisqu'on trouve actuellement des cartes munies d'un 68020 dont l'horloge fonctionne à 25 MHz. Le 68020 se révèle donc utilisable dans des applications déjà complexes.

Une caractéristique importante de ces microprocesseurs est, de notre point de vue, leur aptitude à fonctionner dans un contexte multiprocesseurs. Ils possèdent en effet, des instructions de gestion de sémaphores comme l'instruction TAS (Test And Set) qui, dans un cycle ininterruptible, teste la valeur d'un bit et le positionne à 1 s'il était nul ; le résultat de la comparaison est reporté dans le registre d'état de l'unité centrale.

La cohésion de cette famille de microprocesseurs est assurée d'une part, par la compatibilité ascendante de leurs jeux d'instructions respectifs : le code des plus "vieux" de la famille est exécutable sur les plus "jeunes" (mais il faut retenir que l'inverse n'est pas vrai) et d'autre part, par les nombreux composants qui ont été conçus pour interfacer les unités centrales avec leur environnement. L'utilisation de ces interfaces est très facile puisqu'il suffit dans la plupart des cas de les connecter fil à fil avec le processeur ce qui évite toute la logique de liaison nécessaire auparavant.

On trouve sur le marché, des interfaces de communication série ou parallèle, de gestion de mémoire, d'accès direct mémoire, de gestion de bus etc. Nous nous attarderons plus précisément sur le coprocesseur arithmétique 68881. Pour effectuer des opérations arithmétiques, et particulièrement en virgule flottante, un microprocesseur comme le 68000 ou le 68020 doit exécuter un grand nombre d'instructions. C'est pourquoi MOTOROLA a prévu la conception d'un composant spécialisé avec des instructions spécifiques permettant d'effectuer des calculs rapidement. Utilisé avec un 68020, le 68881 fonctionne en coprocesseur, il prend les données et les instructions directement sur le bus du 68020 et peut exécuter son code en parallèle avec le microprocesseur. Utilisé avec une autre unité centrale de la famille 68000, le 68881 fonctionne en périphérique et pour chaque instruction arithmétique à exécuter, on doit déclencher une interruption sur le microprocesseur, ce qui conduit à un fonctionnement bien moins rapide que précédemment mais néanmoins concurrentiel par rapport au cas où l'on ne dispose pas d'un 68881. Il faut noter que, dans les deux cas précédents, les mécanismes mis en oeuvre sont transparents à l'utilisateur donc faciles d'emploi.

Les composants de la famille 68000 offrent donc des fonctionnalités et des performances qui répondent bien aux critères de rapidité, de souplesse d'emploi et de puissance définis dans le cahier des charges, c'est pourquoi nous les avons retenus pour la réalisation du système d'acquisition.

. 3.2.1. Le bus VME et ses extensions

En parallèle sur le développement du 68000, MOTOROLA a établi une norme qui définit les caractéristiques mécaniques et électriques d'une interconnexion de fond de panier, ainsi que les protocoles d'échange sur cette interconnexion (Annexe 2). Ce bus, baptisé VME (Versa Module Eurocard) a une structure très proche de celle du bus du microprocesseur 68000 et permet d'exploiter au mieux les capacités de ce dernier. C'est un bus rapide qui autorise des débits d'information allant jusqu'à 40 M octets par seconde et offre des fonctionnalités bien adaptées à la gestion de systèmes multiprocesseurs (voir en annexe, dans la description résumée du VME, la gestion hiérarchisée des accès au bus et de la prise en compte des interruptions). Il est de plus, désormais bien implanté dans les milieux scientifiques (CERN notamment) et industriels (contrôle d'automatisme, aviation etc) et on disposera donc sur le marché d'un grand choix de matériels et de logiciels.

Des extensions ont été conçues autour du standard VME afin d'accroître ses performances. Le bus VMX tout d'abord, est une extension locale permettant à une unité centrale d'effectuer des traitements sur des données situées dans une mémoire annexe, sans passer par le VME, ce qui libère ce dernier pour d'autres échanges. Une application très intéressante de l'extension VMX est la réalisation de files d'attentes entre deux unités centrales. Pour ce faire, il suffit d'utiliser une mémoire double accès VME/VMX (Fig. 3.1) : l'une des unités charge la mémoire par VMX tandis que l'autre la vide par VME (ou l'inverse suivant la configuration choisie), on divise ainsi le temps d'occupation du bus VME par deux par rapport au cas où tous les échanges se font sur ce bus.

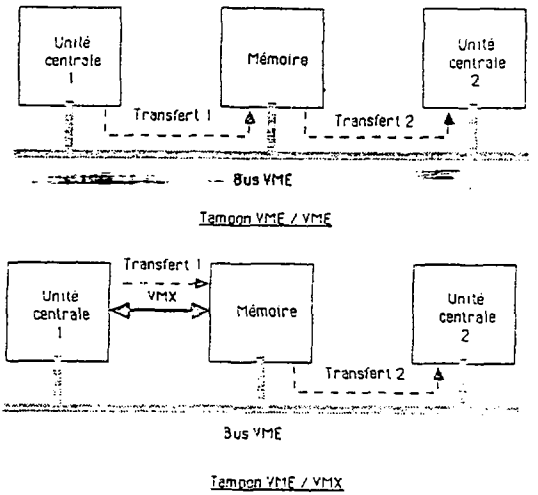


Fig 3.1 : La liaison VMX permet de décharger le bus VME d'un grand nombre de transferts

Par ailleurs, il a été développé au CERN, une liaison parallèle : VMV (VME Vertical) qui permet d'interconnecter plusieurs châssis VME et de communiquer de l'un à l'autre de façon transparente à l'utilisateur (Fig. 3.2). Plus rapide, plus souple d'emploi et plus fiable qu'une liaison série (RS232), cette "branche" est particulièrement intéressante pour télécharger dans des unités centrales cibles, les logiciels qu'elles doivent exécuter. Elle permet en outre, de construire un système de messagerie par "boîte à lettre" entre les différents châssis présents sur l'interconnexion. On peut l'utiliser en troisième lieu pour effectuer des recherches de panne sur des mémoires ou des périphériques situés dans des châssis dépourvus d'outils de test.

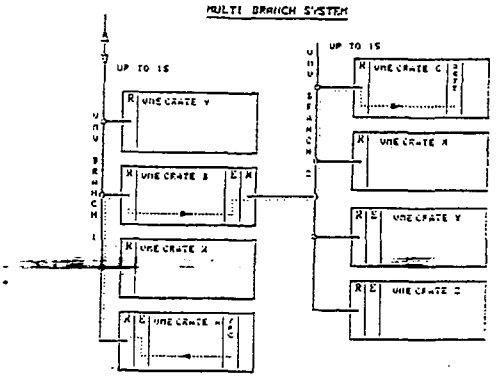


Fig 3.2 : Exemple d'architecture VMV multibranches
Chaque branche VMV peut comprendre jusqu'à 15 châssis.
Les unités centrales du châssis A peuvent accéder aux cartes du châssis C de façon transparente suivant le chemin : Châssis VME A - Branche VMV 1 - Châssis VME B - Branche VMV 2 - Châssis VME C.
Les fonctions émission et réception de données sur la liaison VMV sont assurées par deux modules différents : E (émission) et R (réception).

Le dernier élément que nous signalerons, qui n'est pas à proprement parler une extension du standard VME est un module interface VME/CAMAC (Fig. 3.3). Placé avec un microprocesseur dans un châssis VME, ce module permet de contrôler une branche CAMAC qui apparaît alors à l'unité centrale comme une partie de son champ d'adressage.

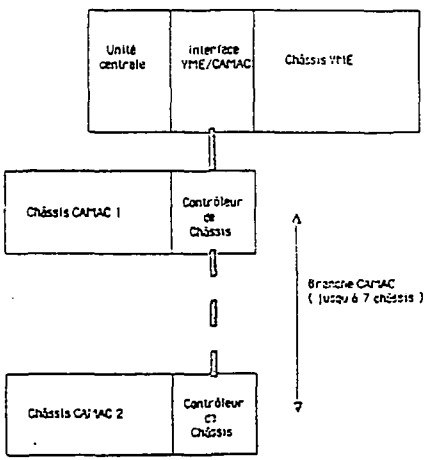


Fig 3.3 : Utilisation de l'interface VME/CAMAC.
La branche CAMAC se comporte comme une fraction de l'espace d'adressage de l'unité centrale située sur le châssis VME.

Compte tenu de sa parenté avec les microprocesseurs de la famille 68000, de ses performances et des possibilités offertes par ses extensions, il était presque "naturel" de choisir le standard VME pour réaliser la partie "intelligente" du système d'acquisition.

3.2. Un exemple d'architecture possible

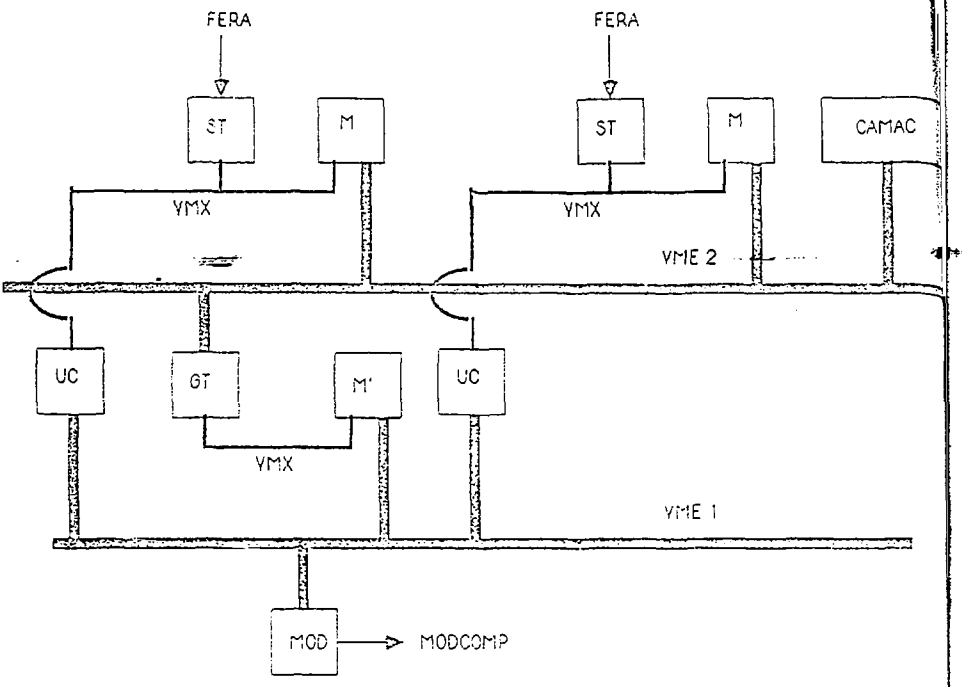
3.2.1. Description générale

Un schéma général de cette architecture est donné figure 3.4. Tous les modules du système ne peuvent être placés sur un même bus VME ; ceci tout d'abord pour une question de place disponible dans un châssis et ensuite pour éviter de saturer ce bus en transferts. On dispose donc d'un premier bus VME (VME1) sur lequel se trouvent les différents dispositifs de lecture et de filtrage associés à chacun des ensembles multidétecteurs. Les sous-événements enregistrés par ces dispositifs sont ensuite regroupés et envoyés au calculateur MODCOMP par l'ensemble GT (Groupement - Transfert) situé sur le bus VME2.

3.2.2. Acquisition - Filtrage

On place en sortie des bus FERA associés à chacun des multidétecteurs, un ensemble permettant de traiter les sous-événements.

Cet ensemble se compose, en premier lieu, d'une carte ST (Sélection - Transfert) réalisant l'interface FERA/VME et organisée en file d'attente circulaire pouvant contenir plusieurs sous-évènements successifs (Fig. 3.5). Les quatre cartes ST correspondant aux différents multidétecteurs, sont synchronisées de telle façon que les sous-événements qui composent l'événement n, se placent tous à un même niveau k dans les files d'attente. Si l'événement courant est rejeté sur décision des cartes OC, aucune action n'est effectuée sur les cartes ST et les données de l'événement suivant (n + 1) viendront écraser les informations déjà présentes dans le tampon k. Si par contre, l'événement courant est accepté, une validation opérée sur les cartes ST incrémente d'une unité, le pointeur d'écriture et c'est le tampon de numéro (k + 1) qui jouera le rôle de tampon d'entrée pour l'événement à venir.



ST : Interface FERA/VME .
M et M' : Cartes mémoire tampon .
GT : Carte de regroupement des sous-événements .
MOD : Carte interface VME/MODCOMP .
UC : Unité centrale de filtrage des sous-événements .

Fig 3.4 : Schéma de principe d'une architecture possible
(on ne représente ici que les ensembles de filtrage de deux multidétecteurs)

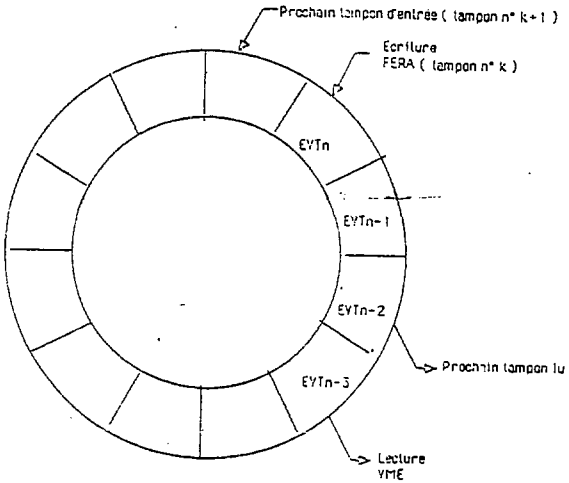


Fig 3.5 : Structure en file d'attente circulaire de la carte ST

Le microprocesseur associé suit le cycle lecture - traitement - écriture suivant. Via le bus VMX, afin de ne pas encombrer le bus VME, il vient saisir le premier sous-événement de la file ST pour le stocker et le traiter dans sa propre mémoire. Les traitements effectués peuvent être de deux types : calcul de nouveaux paramètres d'une part, filtrage de données ou d'événement d'autre part. Dans ce dernier cas, sur la base de critères définis dans les logiciels, l'unité centrale peut décider de l'acceptation ou du rejet du sous-événement qu'elle traite. Si un rejet est décidé, elle le signale à la carte GT qui centralise les décisions des différents processeurs de filtrage et conclut par une acceptation ou un rejet de l'événement global. En cas de rejet, les microprocesseurs de filtrage passent à la lecture d'un nouvel événement qui vient écraser le précédent. Dans le cas contraire, chacune des unités centrales charge, toujours par le bus VMX, le sous-événement courant dans une mémoire tampon M où ce dernier attend d'être lu par l'ensemble GT.

On note qu'ici, la liaison VMX est utilisée comme moyen de communication inter-châssis puisqu'elle relie un microprocesseur situé sur le bus VME1 à une mémoire installée sur le bus VME2. Des essais de connection VMX entre deux châssis ont montré la nécessité de blinder le câble de liaison et de placer à chaque extrémité de ce dernier, un module d'amplification et de mise en forme des signaux. En effet, du fait de la longueur de la liaison (60 cm) et de son emplacement en fond de panier (donc proche des alimentations), on observe des perturbations qui conduisent à des erreurs.

3.2.3. L'ensemble Groupement - Transfert

Cet ensemble est un élément très sensible du système car c'est en ce point que convergent les informations issues des différents multidétecteurs et on doit éviter les phénomènes "d'étranglement" pouvant résulter du passage d'une acquisition parallèle à une acquisition série des données.

Il se compose d'une unité centrale, d'une mémoire tampon et d'une carte interface VME-MODCOMP. L'unité centrale joue deux rôles distincts mais de même priorité. Elle centralise, en premier lieu, les décisions d'acceptation ou de rejet prises par les microprocesseurs de filtrage, prend en fonction de celles-ci une décision sur l'ensemble de l'événement et répercute ensuite cette dernière sur les unités de filtrage. En cas d'acceptation de l'événement, sa seconde fonction est de lire les sous-événements présents dans les différentes mémoires tampon M et de les regrouper sous forme d'un événement complet dans la mémoire M' dont le rôle est d'absorber l'augmentation du flux d'information en ce point (Fig. 3.6). La carte interface VME-MODCOMP, elle, prend les événements en sortie de la file d'attente M' et les envoie vers l'ordinateur en mode DMA.

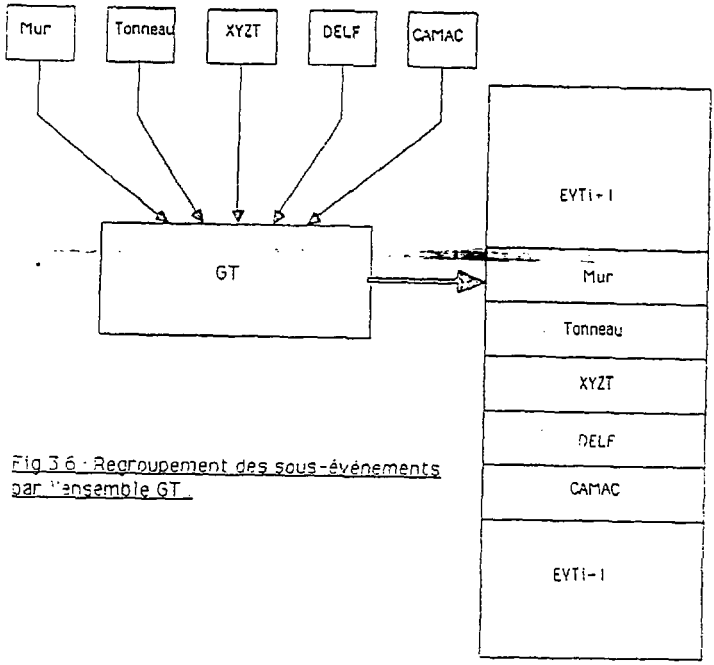


Fig 3.6 : Regroupement des sous-événements par l'ensemble GT

3.2.4. Les problèmes posés par ce type d'organisation

Avec une telle architecture, les multidétecteurs sont autonomes les uns par rapport aux autres et autonomes vis à vis de l'ordinateur MODCOMP mais, pour que le dispositif fonctionne, on doit avoir effectivement une unité centrale par ensemble de détection et si une seule d'entre elles tombe en panne, le système se bloque.

On peut de plus, considérer qu'un tel système n'est pas optimisé car un événement ne peut être pris en compte par l'ensemble GT, que lorsque tous les sous-événements qui le composent ont été traités par les unités de filtrage. Ce dispositif, alignant sa vitesse sur celle du plus lent des processeurs, peut donc conduire à des temps morts importants. De la même façon, l'acquisition reste bloquée au niveau des processeurs de filtrage pendant tout le temps de traitement d'un événement et si ce temps est important par rapport au flux

d'événements en entrée, la capacité des cartes ST se trouvera dépassée et on accusera une perte d'information.

En outre, en phase de test ou lors de recherche de panne avec exécution de logiciel en mode pas à pas, il sera très difficile de reproduire les conditions réelles de synchronisation des différents processeurs.

Ces limitations nous ont paru suffisantes pour abandonner ce modèle d'architecture sans en étudier les détails.

3.3. L'architecture finalement retenue

3.3.1. Description générale

On constate que les problèmes cités au paragraphe 3.2.4. proviennent essentiellement du fait que chacun des dispositifs de filtrage est lié à un ensemble de détection particulier. Il a donc été étudié un schéma d'architecture dans lequel la lecture des événements et leur filtrage sont effectués par des unités centrales indifférenciées tant du point de vue matériel que logiciel. Le résultat de cette réflexion est schématisé figure 3.7. Les données arrivant par le bus FERA transitent en premier lieu par une carte RS (Restructuration - Sélection) qui convertit les données dans un format mieux adapté à une analyse ultérieure que le format FERA. Les sous-événements résultant de cette conversion sont ensuite stockés dans la carte IFV (Interface FERA/VME) assumant à la fois les fonctions de file d'attente et d'interface entre FERA et VME. Les voies CAMAC sont, quant à elles, lues par un microprocesseur muni d'une interface VME/CAMAC. Les microprocesseurs placés en parallèle sur le bus VME2 (Processeur de Filtrage) jouent tous le même rôle de regroupement des différents sous-événements et de filtrage de l'événement résultant. Ils chargent ensuite ce dernier dans leur mémoire tampon MEVi associée (Mémoire Evénement) organisée en file FIFO, où il attend d'être lu et transféré vers l'ordinateur par la carte T (Transfert) qui réalise l'interface VME/MODCOMP [CHA 86].

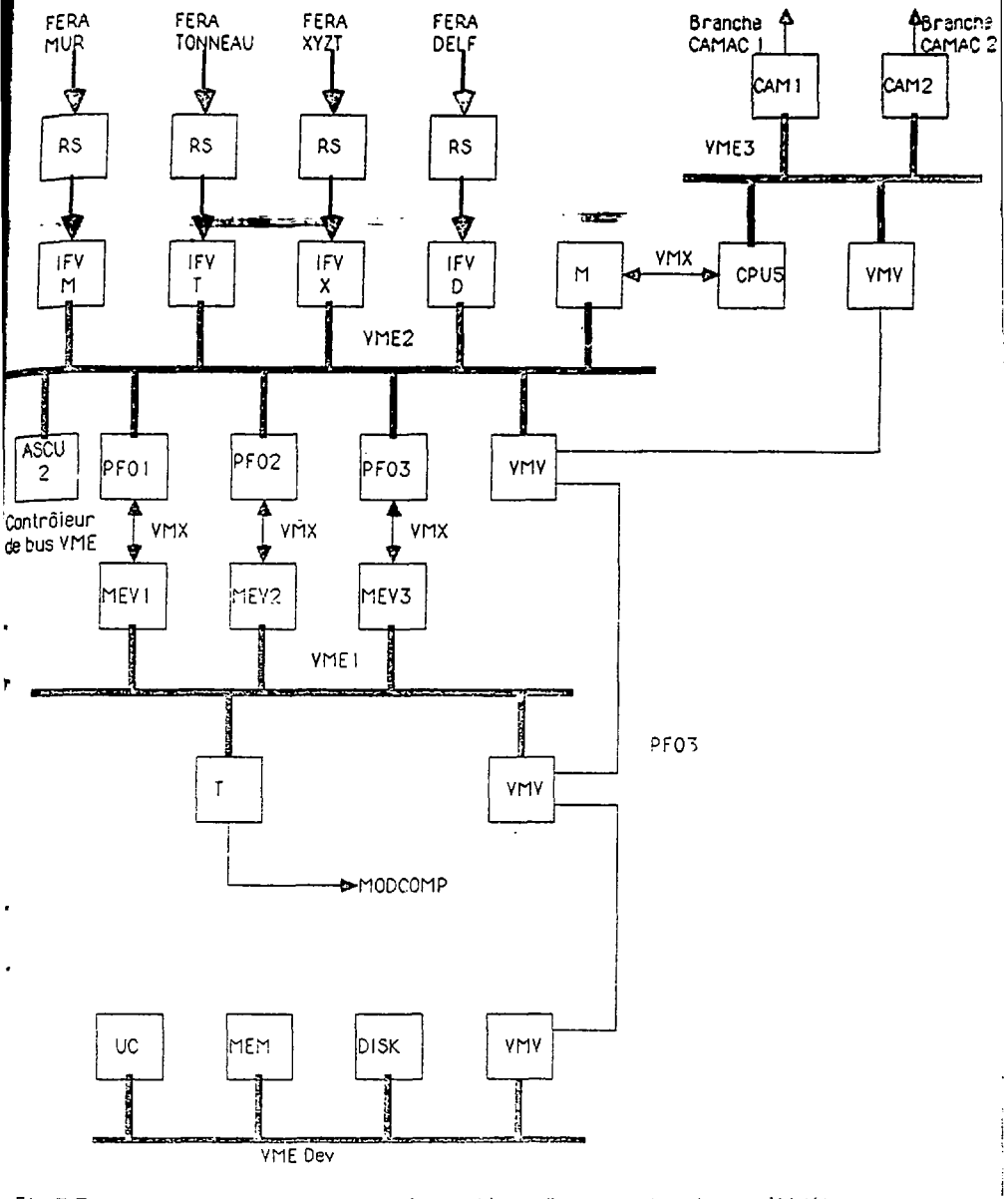


Fig 3.7 : Architecture retenue pour le système d'acquisition des multidétecteurs.

Exempt d'unité centrale maître et des protocoles de gestion et de communication que la présence d'un maître entraine, le système gagne en simplicité et en efficacité.

La gestion et le contrôle des opérations lancées sur tout cet ensemble s'effectue à travers un système de développement. A partir de ce dernier, on peut communiquer avec n'importe quel châssis cible en passant, soit par la liaison VMV, soit par une des liaisons série mises en place (Fig. 3.8). C'est sur le disque dur du système de développement que se trouvent toutes les informations permettant d'initialiser ou d'arrêter une acquisition et de faire, si besoin est, des mises au point sur le dispositif.

3.3.2. L'interface FERA/VME

Le format FERA des données n'est pas facile d'emploi car il faut avant d'utiliser une donnée, filtrer la Sous-Adresse qui lui est associée (Fig. 2.5). En outre, il regroupe les informations par codeur alors qu'il serait plus explicite pour les physiciens de disposer de données réunies par détecteur. Un format spécifique de sous-événement a donc été étudié en collaboration avec les futurs utilisateurs pour chacun des multidétecteurs. Ces formats "sous-événement" ont une structure commune (Fig. 3.9.a) composée d'un en-tête suivi des paramètres des différents détecteurs valides. L'en-tête contient la longueur du sous-événement, son numéro d'apparition, un mot d'état et pour terminer un champ de bit donnant le marquage des détecteurs valides (ces deux derniers éléments sont fournis par la carte DC). La figure 3.9.b indique la structuration des données pour les détecteurs des différents multidétecteurs. On donne le numéro du détecteur ainsi qu'un marquage des voies touchées sur ce dernier. On indique ensuite la valeur obtenue pour chacune des données (les voies non touchées ont une valeur nulle).

Il était prévu, dans un premier temps, que la conversion des données du format FERA vers le format sous-événement soit effectuée par un processus logiciel, mais des essais ont montré que cette dernière

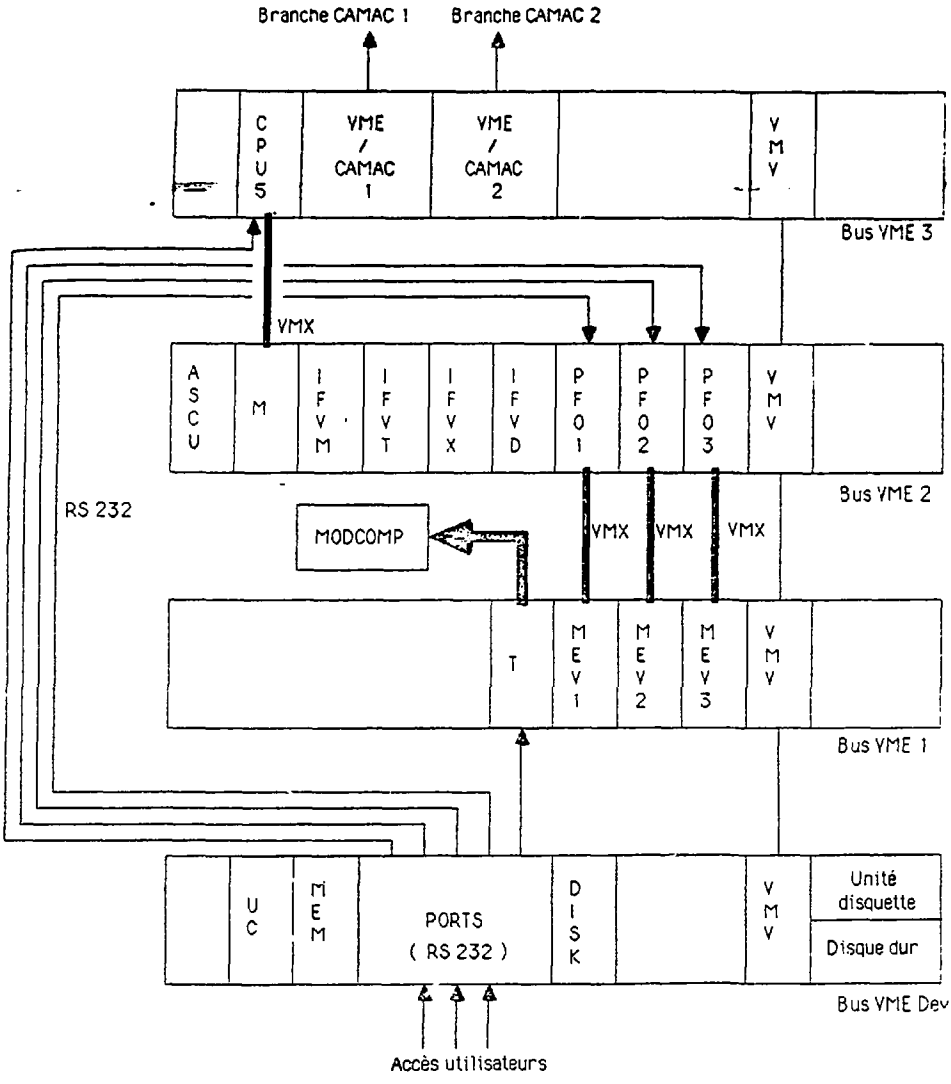
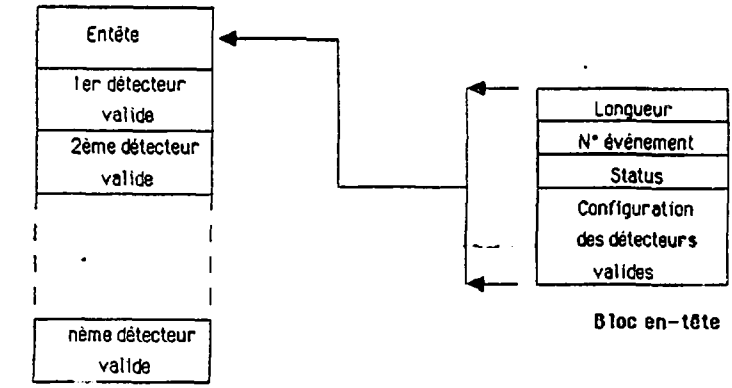
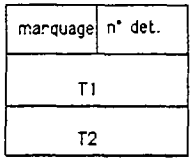


Fig 3.8 : Schéma d'implantation des modules VME dans les différents châssis .

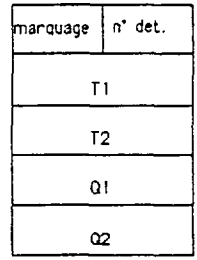


Structure d'un sous-événement lié à un multidétecteur .

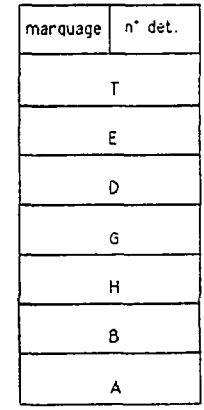
Fig 3.9.a : Structure générale du format sous-événement .



Format d'un bloc détecteur MUR



Format d'un bloc détecteur TONNEAU



Format d'un bloc détecteur XYZT ou DELF

Fig 3.9.b : Structuration des données pour chaque type de détecteur .

demande alors un minimum de 6 à 7 µs par donnée ce qui se révèle prohibitif pour notre application. En effet, si l'on ajoute à ces 7 µs (déjà presque le double du temps mort autorisé par le cahier des charges), le temps de traitement des données proprement dit, on atteint rapidement des valeurs de temps mort supérieures à ce que le filtrage, avec la réduction de données qui en résulte, peut compenser.

Il s'est donc révélé nécessaire de concevoir une carte (RS) qui opère cette conversion en logique cablée. Cette carte (Fig. 3.10) reçoit en entrée les données issues des codeurs et grâce à une table de

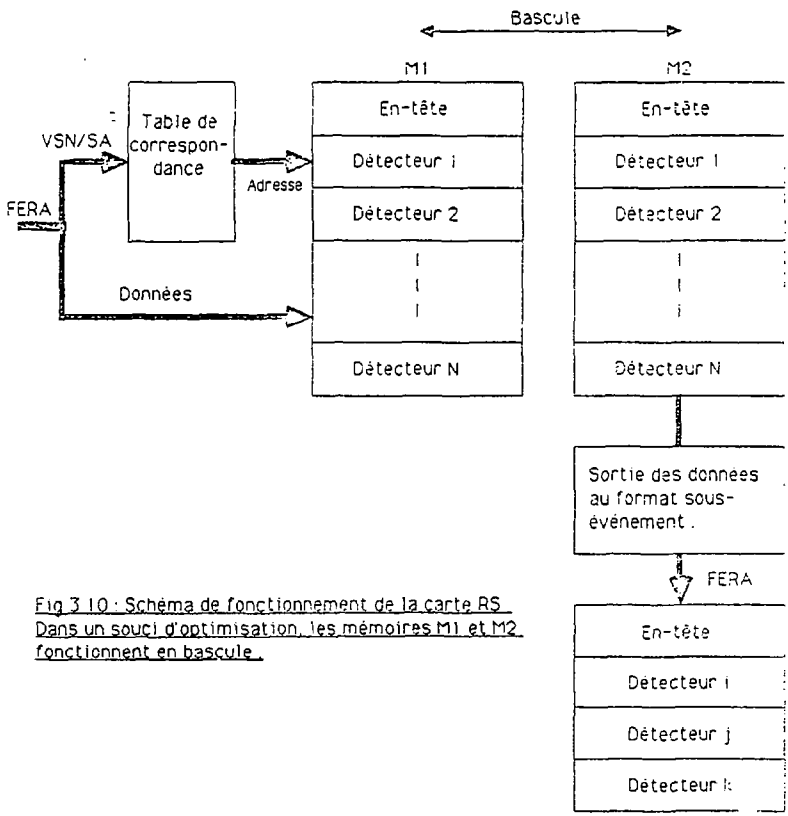
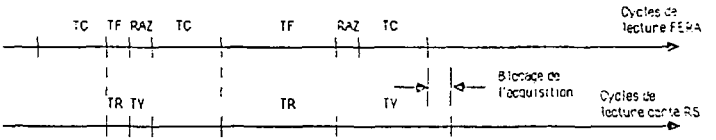


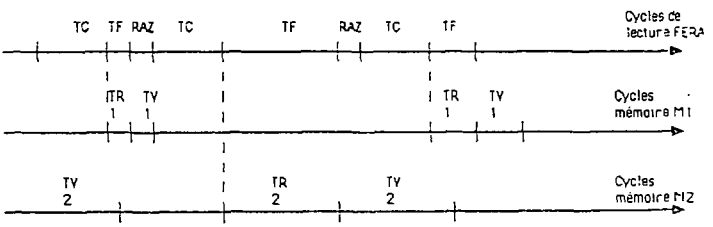
Fig 3.10 : Schéma de fonctionnement de la carte RS
Dans un souci d'optimisation, les mémoires M1 et M2 fonctionnent en bascule.

correspondance, qui décode le couple (VSN,SA), elle les positionne à une place fixe dans la mémoire M_1. On obtient ainsi un sous-événement (dont l'organisation est fonction du multidétecteur) de structure fixe et dans lequel toutes les voies de tous les détecteurs sont représentées. Pendant ce temps, le sous-événement de la génération précédente, stocké dans M_2, est émis vers la carte IFV sur un bus FERA avec un format sous-événement des données. Lorsque M_1 a été remplie et M_2 vidée, on effectue une remise à zéro de M_2 et on échange les rôles de ces deux mémoires. Ce fonctionnement en bascule améliore les performances du système par rapport au cas où l'on aurait qu'une seule mémoire. En effet, dans ce dernier cas, on aurait le type de chronogramme 1(Fig. 3.11) qui fait apparaître que l'acquisition d'un événement "long" peut conduire à un blocage de l'acquisition. Le chronogramme 2 (Fig. 3.11) montre que l'utilisation de deux mémoires en bascule permet de prendre en compte, un événement même si le précédent n'est pas encore complètement enregistré.



TC : Temps de codage.
TF : Temps de lecture FERA des événements.
RAZ : Temps de remise à zéro des codeurs.
TR : Temps de remplissage du tampon d'entrée.
TV : Temps de vidage du tampon d'entrée.

Chronogramme 1 : Utilisation d'un seul tampon en entrée



TR1 : Temps de remplissage mémoire M1.
TR2 : Temps de remplissage mémoire M2.
TV1 : Temps de vidage mémoire M1.
TV2 : Temps de vidage mémoire M2.

Chronogramme 2 : Utilisation de deux tampons en entrée.

Fig 3.11 : Avec deux mémoires fonctionnant en bascule (chrono 2), on peut éviter des blocages de l'acquisition qui se produiraient si l'on n'avait qu'un seul tampon en entrée (chrono 1).

En ce qui concerne les voies additionnelles qui seront lues par FERA, aucune structure particulière de donnée n'a encore été définie et on ne dispose pas d'une carte de marquage générale permettant de fournir une configuration de bit exploitable par une éventuelle carte RS. On devra donc utiliser ces données directement au format FERA.

La carte IFV joue pour sa part le même rôle que la carte ST dans l'architecture décrite au paragraphe 3.2, elle reçoit les sous-événements en sortie de la carte RS et les stocke successivement à la queue d'une file d'attente. Cette carte est une carte d'usage général qui peut être utilisée pour toute expérience mêlant les standards FERA et VME.

3.3.3. La lecture du CAMAC

Le schéma donné figure 3.7 montre que l'on prévoit l'implantation de deux branches CAMAC dans le système. Ces branches interviennent en premier lieu, lors de l'initialisation des modules CAMAC. Il s'agit par exemple, de positionner des seuils sur les codeurs, de programmer la valeur de leur VSN ou leur mode de fonctionnement sur le bus FERA mais aussi de charger les tables de décision dans les modules du configurateur ou dans les cartes OC. La rapidité d'exécution n'est pas un critère important en phase d'initialisation et le standard CAMAC reste alors bien adapté pour effectuer ces opérations. Mais on utilise aussi ce standard pour lire les modules ne disposant pas d'un bus FERA en face avant, et cette fois par contre, une rapidité maximum est requise.

La gestion des modules CAMAC ne présente pas un niveau de complexité justifiant l'usage d'un microprocesseur 68020. On utilise donc, pour ses performances en rapidité, un microprocesseur 68010 monté sur une carte CPU5 de FORCE COMPUTERS [FORC]. Sur le même bus VME, on place deux interfaces VME/CAMAC (CAM1 et CAM2) [CES] contrôlant chacun une des deux branches mises en oeuvre. L'unité centrale déclenche l'exécution d'un ordre CAMAC en fournissant à ces modules, un codage approprié sur les bus d'adresses et de données.

Une telle organisation permet avec un 68000, d'exécuter des instructions CAMAC en environ 4 µs, on peut donc espérer améliorer ces résultats avec l'utilisation d'un 68010. On remarque que le temps d'exécution d'un ordre CAMAC varie énormément en fonction du mode d'adressage utilisé par l'unité centrale pour activer la branche. On trouve en effet 5,5 µs pour une lecture en adressage direct contre 3,5 µs pour un accès indexé sur un registre [POS 85]. Cet élément devra être pris en ligne de compte pour l'optimisation des logiciels d'acquisition.

En cours d'exécution de ces derniers, les données lues par CAMAC seront stockées, via un bus VMX, dans une mémoire tampon organisée en file d'attente et jouant pour les voies CAMAC le rôle que les cartes IFV jouent pour les voies FERA.

3.3.4. Filtrage des événements

Toutes les unités centrales placées sur le bus VME2 assument une même fonction qui se décompose en trois points : regroupement des sous-événements, traitement de l'événement ainsi constitué et enfin stockage des résultats dans une mémoire tampon. Un processus à la fois logiciel et matériel (paragraphe 4.3) permet de désigner un microprocesseur, PF_i par exemple, pour l'acquisition d'un événement incident. Ce microprocesseur lit successivement les données contenues dans les cartes IFV et la carte M pour les regrouper sous forme d'un événement complet dans sa mémoire locale. Dès la fin de ce premier travail, un autre processeur PF_j peut se trouver désigné pour l'acquisition d'un nouvel événement tandis que PF_i exécute un traitement des informations qu'il vient de mémoriser. Ce traitement, qui peut être standard ou conçu par l'utilisateur débouche sur une acceptation ou un rejet de l'événement. En cas de rejet, PF_i peut se remettre immédiatement en attente d'une nouvelle acquisition ; en cas d'acceptation, au contraire, il doit d'abord placer l'événement traité en file d'attente dans une mémoire tampon accessible par VMX.

Un premier point remarquable de cette architecture est qu'elle peut fonctionner avec à la limite, un seul processeur PF. Donc si une panne immobilise un des microprocesseurs d'acquisition, la vitesse d'acquisition diminue mais le système ne reste pas bloqué contrairement à ce qui se passait avec l'architecture décrite au paragraphe 3.2. Par ailleurs, on peut prévoir de conditionner le type de traitement subi par un événement sur configuration de bits ou sur fenêtre par exemple. Le temps d'exécution d'un traitement dépendra donc de la longueur et du type d'événement considéré et il faudra doser la complexité des traitements effectués de façon que le flux de données en sortie soit compatible avec le temps mort demandé dans le cahier des charges. Toutefois, l'architecture présentée ici permet de réaliser des traitements complexes et longs sur certains types d'événements rares. En effet, le traitement d'un événement s'effectuant en local sur l'unité centrale désignée pour la prise en compte de ce dernier, les autres processeurs PF peuvent, pendant ce temps, continuer l'acquisition. Dans ces conditions, on peut perdre l'historique des événements, c'est-à-dire l'ordre de leur apparition dans le temps mais ceci ne pose pas de problème pour les traitements ultérieurs.

3.3.5. Transfert vers l'ordinateur

La carte T située sur le bus VMEI lit les événements successifs dans les cartes MEVi pour les tranférer vers l'ordinateur MODCOMP. La distance à parcourir étant de plusieurs dizaines de mètres, les transmissions s'effectuent sur une liaison différentielle. Construite au GANIL, cette carte est munie d'un microprocesseur 68010 accompagné d'un interface DMA permettant de communiquer des données au MODCOMP à la vitesse de 250 k mots par seconde (vitesse maximale autorisée par le canal DMA du calculateur).

3.3.6. Contrôle et visualisation

Donnant la priorité au développement de l'acquisition, ce point a été jusqu'à présent peu étudié. Toutefois, on s'est attaché à rester ouvert quant aux possibilités de contrôle et de visualisation sur le système.

Une première possibilité consiste à utiliser, pour le contrôle individuel des différents multidétecteurs, une carte graphique du type de la carte Z développée pour le système d'acquisition du Château de Cristal [VIL 84]. Ce module au standard VME, placé en position d'espion sur le bus FERA décode les couples (VSN, SA) et peut ainsi incrémenter des spectres liés à des détecteurs déterminés. Cependant, on peut aussi placer un ou plusieurs ensembles graphiques (Unité Centrale + interface graphique) sur le bus VME 1 qui traiteraient les événements après filtrage. Signalons qu'il serait difficile de placer une telle carte sur le bus VME 2 qui sera déjà très chargé en transfert. La troisième solution, peut-être la plus "moderne", serait d'installer une carte interface entre le système au standard VME et une station de travail dont les possibilités de traitement et de calcul seraient bien plus étendues que celles d'une simple carte graphique.

Ne disposant, pour l'instant d'aucune de ces solutions, le contrôle des premières expériences sera effectué par l'ordinateur MODCOMP.

CHAPITRE 4

LES LOGICIELS DU SYSTEME D'ACQUISITION

Ce chapitre présente les problèmes logiciels rencontrés pour la mise en oeuvre du système d'acquisition et les solutions proposées. Nous décrirons en premier lieu, les éléments du choix d'un système d'exploitation et de langages de programmation adaptés aux critères énoncés dans le cahier des charges : rapidité d'exécution et souplesse d'emploi. Nous exposerons ensuite les solutions apportées aux problèmes posés par le caractère multiprocesseurs de notre architecture. Les unités centrales cibles ne disposant pas d'unité disque, nous avons dû développer des utilitaires de communication permettant de leur fournir les logiciels à exécuter. Nous avons de plus, pour des raisons de facilité d'emploi, prévu un dispositif logiciel d'initialisation automatique de l'ensemble du système d'acquisition. De même, nous avons défini un système de gestion de mémoire tenant compte du partage de celle-ci entre différents processeurs. Nous décrirons, pour terminer, l'algorithme de synchronisation des processeurs PF de filtrage des données.

4.1. Choix d'un système d'exploitation et de langages de programmation

4.1.1. Le système d'exploitation

Pour faciliter l'utilisation des microprocesseurs, on implante un système d'exploitation dans leurs mémoires locales respectives. Ce dernier permet, entre autres, une gestion aisée des différentes tâches et offre des utilitaires de recherche de panne (affichage du contenu de registres ou de mémoires, exécution de programme en mode pas à pas etc). Cependant, pour des raisons de rapidité, ce système d'exploitation doit être "temps réel" c'est-à-dire qu'il doit n'augmenter qu'un minimum, le temps de réaction de l'unité centrale à un événement extérieur (une interruption par exemple), par rapport au fonctionnement purement matériel de cette dernière.

La carte CPU 20 retenue pour la réalisation des processeurs PF, supporte un seul système d'exploitation temps réel, il s'agit de PDOS développé par la firme EYRING [EYRI]. Dans un souci d'homogénéité, c'est ce système que nous implanterons sur toutes les unités centrales cibles du dispositif d'acquisition. Peu incombrant (son noyau n'occupe que 6 k octets en mémoire), PDOS est un système temps réel, multitâches, et multiutilisateurs [CAR 86]. En outre, il est d'un apprentissage simple et se révèle, par les fonctionnalités qu'il offre, utilisable à la fois pour les développements de logiciels et pour la mise en place d'applications cibles. Il permet en particulier à l'utilisateur d'organiser la gestion des tâches à sa convenance grâce à des mécanismes de communication appelés "événements" qui influent sur la gestion du temps partagé effectuée par le noyau. Un événement au sens PDOS du terme, est un drapeau à deux états. Une tâche active à un instant donné peut se mettre en attente du positionnement d'un événement par une autre tâche, un programme d'interruption ou après un délai. Il faut noter qu'afin d'obtenir un temps de réponse minimum, les interruptions ne sont jamais inhibées sous PDOS, ce qui permet de les servir très rapidement. De plus, les événements étant communs à toutes les tâches, ils peuvent aussi jouer un rôle de sémaphore dans la synchronisation des travaux effectués par celles-ci. Par ailleurs, on dispose de points d'entrée sur les logiciels de gestion des disques et des ports de communication. Ces "BIOS" (Basic Input Ouput System) peuvent être adaptés voire complètement réécrits en fonction des besoins de l'utilisateur.

PDOS offre de plus une fonctionnalité intéressante : la "RAM DISK". On peut en effet, par une commande demander au noyau de considérer et de traiter toute une zone de mémoire comme un disque sur lequel on stockera les fichiers voulus. Sur le système de développement, la RAM DISK contient les bibliothèques et les utilitaires de compilation ce qui permet d'exécuter cette dernière beaucoup plus rapidement que lorsqu'on ne dispose que d'un disque dur ordinaire. Par ailleurs, lors du téléchargement et de l'utilisation des logiciels dans les microprocesseurs cibles, la RAM DISK se révèlera un outil important.

Il est à noter que le système de développement utilisé actuellement fonctionne sous PDOS mais que l'on envisage, dès que possible, de passer à un système UNIX afin de disposer d'utilitaires plus évolués et surtout de meilleures protections sur les accès aux ressources.

4.1.2. Les langages de programmation

Trois langages seront fournis pour l'écriture des logiciels : Assembleur 68000/68020, FORTRAN et C.

L'assembleur ne sera utilisé que pour l'écriture de programmes proches du système d'exploitation ou demandant une grande vitesse d'exécution. Son usage sera donc, en pratique, réservé aux spécialistes.

On prévoit par ailleurs, pour l'ensemble des utilisateurs d'implanter des langages évolués dont l'utilisation est beaucoup plus simple que celle de l'assembleur. On retient tout d'abord le FORTRAN car c'est un langage désormais "rôdé", bien implanté dans les laboratoires et bien connu des physiciens. Mais on retient surtout le langage C qui est à la fois structuré, performant, simple d'emploi et facile d'apprentissage [KER 84]. Le formalisme d'écriture des tests et des boucles conduit de lui-même à une structuration des programmes et, de plus, l'interdiction d'utiliser des variables non déclarées au préalable oblige le "programmeur C" à une discipline qui permet d'éviter des erreurs. Autre point fort de ce langage, il permet l'utilisation de pointeurs et de structures. Un pointeur permet de traiter une entité par l'intermédiaire de l'adresse de celle-ci et "une structure est un ensemble comprenant une ou plusieurs variables, parfois de types différents, qu'on regroupe sous un seul nom pour faciliter le traitement. (Les structures sont baptisées "records" dans d'autres langages comme en PASCAL)" [KER 84]. Ce type de variable sera particulièrement intéressant pour définir les descripteurs utilisés dans les logiciels. On voit donc que l'on aura intérêt à utiliser le langage C autant que possible dans l'écriture des programmes d'acquisition.

4.2. Lancement d'une application

Nous appellerons ici "application" : la réunion d'un ensemble matériel et des logiciels devant être exécutés par ce dernier.

4.2.1. Système de description d'une application

La nécessité d'un descripteur d'application nous est apparue en premier lieu, lors d'essais de communication avec les microprocesseurs cibles. Sous POOS, en effet, il faut, pour accéder à ces derniers, connaître les numéros des différents ports de communication auxquels ils sont respectivement liés sur le système de développement. On s'aperçoit rapidement que ce mode d'accès est fastidieux et source d'erreurs et qu'il serait préférable de reconnaître les différentes unités centrales par un nom significatif de leur fonction. Un fichier de correspondance a donc été créé, permettant d'associer à chacune d'elles, un nom et un numéro de port et c'est ce nom que l'on doit alors utiliser en argument des utilitaires de communication, pour désigner l'interlocuteur désiré.

Ce procédé résoud bien les problèmes d'accès aux différents éléments de l'ensemble du système mais on s'aperçoit que la notion de description d'application peut devenir plus générale lorsqu'on aborde les questions d'initialisation et de lancement de l'acquisition. En effet, pour que le système devienne opérationnel, le microprocesseur situé sur le châssis de développement doit fournir aux unités centrales cibles, les logiciels qu'elles devront exécuter et les données de base nécessaire à ce travail. On conçoit bien que, dans le contexte multiprocesseurs envisagé, ces opérations ne soient pas simples à réaliser "manuellement", c'est pourquoi on s'oriente vers la mise en oeuvre d'un dispositif logiciel dont le rôle sera d'initialiser automatiquement tous les paramètres nécessaires au démarrage de l'acquisition.

De plus, afin d'être utilisable pour différentes configurations matérielles ou logicielles, ce descripteur doit être évolutif, modulaire et rester, dans sa conception, indépendant de toute application particulière.

Il se compose de deux fichiers, l'un décrivant l'application et l'autre indiquant les actions à exécuter pour le lancement de celle-ci.

Un exemple de fichier description d'application est donné figure 4.1. Les éléments fonctionnels du système sont répertoriés par rubriques : UC pour les unités centrales, MEM pour les mémoires et COM pour les chemins de communication. A un deuxième niveau, on trouve une sous-rubrique par élément à décrire. On attribue alors à ce dernier, un nom à la fois significatif de sa fonction et explicite pour l'utilisateur ; c'est par ce nom que l'on désigne l'élément dans le reste du fichier et dans les utilitaires de communication. Toutes les rubriques et sous-rubriques commencent par le caractère "> " et se terminent par le caractère "<". On encadre les commentaires de la même façon que dans le langage C par les séparateurs "/*" et "*/". Les données sont repérées grâce à un jeu de mots clés (soulignés dans l'exemple) que l'on pourra étendre en fonction de nouvelles rubriques à créer. De plus, dans le cas où plusieurs cartes présentent des caractéristiques communes (comme les processeurs PFi dans notre architecture), ces dernières peuvent être regroupées dans un tronc commun auquel on attribue un type (PF ici) par lequel on le référence dans la description des cartes concernées grâce au mot clé INS (Insertion).

On définit d'autre part, les chemins qui permettent, à partir du système de développement, d'accéder à différentes cartes pour leur fournir des commandes ou des données. Pour une unité centrale, le chemin d'accès se décompose en un chemin de données (CD) par lequel transitent les logiciels à télécharger par exemple, et un chemin de commandes (CC) par lequel on transmet des commandes au microprocesseur. Pour une carte

```
> UC                                  /* Description des unites centrales */

    > PF  INS                         /* Tronc commun de description des
                                         processeurs PF */
          CD  CAN1                    /* Chemin de passage des donnees */
          SE  GF2000S, $ADchargement  /* Nom du syst. d'exploitation et
                                         adresse de chargement */
          BP  COMM
    <
    > PF1
          INS  PF                     /* Tronc commun PF */
          CC  LIN 2                   /* Chemin de passage des commandes */
          SE  , , $ADlancement1       /* Adresse de lancement du SE */
          BP  EVT_FERA1               /* Stockage des evt. traites par PF1 */
    <
    > PF2
          INS  PF
          CC  LIN3
          SE  , , $ADlancement2
          BP  EVT_FERA2
    <
    > CPU5
          CD  LIN3
          CC  LIN3
          SE  GF5000S , $ADchargement , $ADlancement
          BP  EVT_CAM
    <
> MEM                                 /* Description du decoupage memoire
                                         initial */

          > EVT_CAM  BP               /* Stockage sous-evenement CAMAC */
                     AD(VME) $EVT_CAM_DEB , $EVT_CAM_FIN
          <
          > COMM     BP               /* Tampon de communication */
                     AD(VME) $COMM_DEB , $COM_FIN
          <
          > RDPF     RD               /* RAM DISK commune a deux chassis
                                         pour telechargement */
                     AD(VME) $RDPF_DEB , Taille , Numero logique
          <
          > EVT_FERA1 BP              /* Stockage event. traites par PF1 */
                     AD(VME) $MEV1_DEB , $MEV1_FIN
          <
          > EVT_FERA2 BP              /* Stockage event. traites par PF2 */
                     AD(VME) $MEV2_DEB , $MEV2_FIN
          <
> COM                                 /* Description des chemins d'acces aux
                                         cartes cibles . */
    > CAN1  RD  RDPF                  /* RAM DISK commune a deux chassis pour
                                         les telechargements . */
    <
    > LIN1  RS232  6                  /* Numeros des ports de communication
                                         serie par lesquels on peut converser
                                         avec les processeurs cibles */
    <
    > LIN2  RS232  7
    <
    > LIN3  RS232  8
    <
```

FIG 4.1 : Exemple de fichier de description d'application .
(On ne decrit pas ici le fichier complet de description du système)

"passive" (sans unité centrale), on donne uniquement un chemin de données sauf si le système de développement n'a pas directement accès à la carte, auquel cas, il faut indiquer une unité centrale "relais" à laquelle on passe les opérations à exécuter sur la carte visée. Les chemins d'accès sont détaillés dans la rubrique COM. Un chemin peut être une ligne série RS232 (RS232), mais peut aussi être une mémoire (M) ou une RAM DISK (RD) accessible à la fois par l'unité cible et le système de développement (paragraphe 4.2.2).

Détaillons maintenant les spécificités de chaque rubrique. Pour une unité centrale, on précise le nom du fichier contenant le système d'exploitation (SE) ainsi que les adresses de chargement et de lancement de ce dernier. On indique aussi, les blocs de mémoire auquels le processeur peut accéder (BP : bloc primaire). Dans la rubrique MEM, on spécifie le découpage de la mémoire (paragraphe 4.2.3) en blocs primaires et éventuellement en blocs secondaires (BP et BS) ainsi que les RAM DISK (RD) à initialiser.

Signalons que seuls les paramètres configurables, de façon matérielle ou logicielle, des cartes sont spécifiés dans le fichier de description. On dispose en effet, d'un ensemble de fichiers de constantes indiquant chacun les caractéristiques fixes d'un type de carte ou de composant. La figure 4.2 illustre l'utilisation de ces fichiers pour une carte contrôleur de bus VME ASCU2 (Advanced System Controller Unit) de FORCE COMPUTERS : On donne dans le fichier ASCU2$P1:H, les déplacements par rapport à l'adresse de base de la carte, des différents composants qu'elle comporte puis on donne dans MPCC$P1:H, PI-T$P1:H, RTC$P1:H et BIM$P1:H les déplacements des registres de chaque composant par rapport à l'adresse d'implantation de ce dernier.

C'est le fichier descriptif de l'application qui fournit tous les paramètres par défaut pour l'exécution des commandes décrites dans le fichier de lancement de l'application. Ces dernières sont choisies

```
/*
        ASCU2P1:H -- last update : 28 jan 87     creation date : 15 jan 87
        Copyright Acquisition group, GANIL Caen, France
        author  : Herve POSTEC

        contents : ASCU2 board hardware description .

        List of the files to be included before the present one : BEFore FILE .
        ( These files must necessary be included in the application )

        BEFILE:         -> None .
                        ENDBE

        List of the include files to be used with the present one : WITH FILE .
        ( Part or all of these file may be included or not in the
          application depending on the aimed purpose . If included, their
          relative location to the present file does not mind )

        WITHFILE:       -> MPCC2P1:H    MPCC registers layout
                        -> PI_T2P1:H    PI_T registers layout
                        -> RTC2P1:H     RTC registers layout
                        -> BIM2P1:H     BIM registers layout
                        ENDWI

*/

/* Define ASCU2 chips offsets */

#define ASC2MPCC        0x00L   /* MPCC BASE OFFSET */
#define ASC2PIT1        0x40L   /* PI/T1 BASE OFFSET */
#define ASC2PIT2        0x80L   /* PI/T2 BASE OFFSET */
#define ASC2RTC         0xC0L   /* RTC BASE OFFSET */
#define ASC2BIM1        0x100L  /* BIM1 BASE OFFSET */
#define ASC2BIM2        0x110L  /* BIM2 BASE OFFSET */
#define ASC2BIM3        0x120L  /* BIM3 BASE OFFSET */
#define ASC2BIM4        0x130L  /* BIM4 BASE OFFSET */


/*
        BIM2P1:H -- last update : 28 jan 87     creation date : 9 jan 87
        Copyright Acquisition group, GANIL Caen, France
        author  : Herve POSTEC

        contents : BIM 68153 registers layout for ASCU2 and CPU5 .

        List of the files to be included before the present one : BEFore FILE .
        ( These files must necessary be included in the application )

        BEFILE:         -> None .
                        ENDBE

        List of the include files to be used with the present one : WITH FILE .
        ( Part or all of these file may be included or not in the
          application depending on the aimed purpose . If included, their
          relative location to the present file does not mind )

        WITHFILE:       -> None .
                        ENDWI

*/
/* 68153 Register offset for ASCU2 and CPU5 :

BIMCR : BIM interrupt control register .
BIMVR : BIM interrupt vector register .

*/

#define BIMCRD_1        0x1L
#define BIMCRI_1        0x3L
#define BIMIRD_1        0x5L
#define BIMIRC_1        0x7L
#define BIMVRD_1        0x9L
#define BIMVRI_1        0xBL
#define BIMVRC_1        0xDL
#define BIMVRD_1        0xFL
```

```
/*
        PI_T2P1:H -- last update : 28 jan 87     creation date : 9 jan 87
        Copyright Acquisition group, GANIL Caen, France
        author  : Herve POSTEC

        contents : PI_T registers layout  for ASCU2 and CPU5 boards .

        List of the files to be included before the present one : BEFo
        ( These files must necessary be included in the application )

        BEFILE:         -> None .
                        ENDBE

        List of the include files to be used with the present one : WIT
        ( Part or all of these file may be included or not in the
          application depending on the aimed purpose . If included, the
          relative location to the present file does not mind )

        WITHFILE:       -> None .
                        ENDWI

*/

/* 68230 PI_T registers offset for ASCU2 and CPU5 */

#define PGCR_1          0x01L           /* PORT GENERAL CONTROL REG */
#define PSRR_1          0x03L           /* PORT SERVICE REQ REG */
#define PADDR_1         0x05L           /* PORT A DATA DIRECTION REG */
#define PBDDR_1         0x07L           /* PORT B DATA DIRECTION REG */
#define PCDDR_1         0x09L           /* PORT C DATA DIRECTION REG */
#define PIVR_1          0x0BL           /* PORT INT VECTOR REG */
#define PACR_1          0x0DL           /* PORT A CONTROL REG */
#define PBCR_1          0x0FL           /* PORT B CONTROL REG */
#define PADR_1          0x11L           /* PORT A DATA REG */
#define PBDR_1          0x13L           /* PORT B DATA REG */
#define PAAR_1          0x15L           /* PORT A ALTERNATE REG */
#define PBAR_1          0x17L           /* PORT B ALTERNATE REG */
#define PCDR_1          0x19L           /* PORT C DATA REG */
#define PSR_1           0x1BL           /* PORT STATUS REG */
#define TMCR_1          0x21L           /* TIMER CONTROL REG */
#define TIVR_1          0x23L           /* TIMER INT VECTOR REG */
#define CPRH_1          0x27L           /* COUNTER PRELOAD REG */
#define CPRM_1          0x29L
#define CPRL_1          0x2BL
#define CNTRH_1         0x2FL           /* COUNTER REG */
#define CNTRM_1         0x31L
#define CNTRL_1         0x33L
#define TMSR_1          0x35L           /* TIMER STATUS REG */
```

Fig 4.2 : Organisation des fichiers de description des constantes de la carte ASCU2 de FORCE COMPUTERS et des composant qu'elle utilise

```
MPCC68P1:H -- last update : 28 jan 87    creation date : 9 jan 87
Copyright Acquisition group, GANIL Caen, France
author  : Herve POSTEC

contents : 68560, 68561 MPCC register layout for ASCUC and CPU5 .

List of the files to be included before the present one : BEFore FILE .
( These files must necessary be included in the application )

BEFILE:         -> None .
                ENDBE

List of the include files to be used with the present one : WITH FILE .
( Part or all of these file may be included or not in the
  application depending on the aimed purpose . If included, their
  relative location to the present file does not mind )

WITHFILE:       -> None .
                ENDWI
*/

/* 68560, 68561 registers offset for ASCUC and CPU5 */

#define RSR_1       0x1L    /* Receiver Status Reg */
#define RCR_1       0x2L    /* Receiver Control Reg */
#define RDR_1       0x3L    /* Receiver Data Reg */
#define RIVNR_1     0x5L    /* Receiver Int Vect Number Reg */
#define RIER_1      0x6L    /* Receiver Int Enable Reg */
#define TSR_1       0x9L    /* Transmitter Status Reg */
#define TCR_1       0xAL    /* Transmitter Control Reg */
#define TDR_1       0xBL    /* Transmitter Data Reg */
#define TIVNR_1     0xDL    /* Transmitter Int Vect Number Reg */
#define TIER_1      0xEL    /* Transmitter Int Enable Reg */
#define SISR_1      0x11L   /* Serial Interface Status Reg */
#define SICR_1      0x12L   /* Serial Interface Control Reg */
#define SIVNR_1     0x15L   /* Serial Interf Int Vect Number Reg */
#define SIER_1      0x16L   /* Serial Interface Int Enable Reg */
#define PSR1_1      0x19L   /* Protocol Select Reg 1 */
#define PSR2_1      0x1AL   /* Protocol Select Reg 2 */
#define AR1_1       0x1DL   /* Address Reg 1 */
#define AR2_1       0x1EL   /* Address Reg 2 */
#define BRGR1_1     0x20L   /* Baud Rate Divider Reg 1 */
#define BRGR2_1     0x21L   /* Baud rate Divider Reg 2 */
#define CCR_1       0x24L   /* Clock Control Reg */
#define ECR_1       0x25L   /* Error Control Reg */
```

```
RTCCP1:H -- last update : 28 jan 87    creation date : 15 jan 87
Copyright Acquisition group, GANIL Caen, France
author  : Herve POSTEC

contents : M58167A RTC register layout for ASCUC board .

List of the files to be included before the present one : BEFore FILE .
( These files must necessary be included in the application )

BEFILE:         -> None .
                ENDBE

List of the include files to be used with the present one : WITH FILE .
( Part or all of these file may be included or not in the
  application depending on the aimed purpose . If included, their
  relative location to the present file does not mind )

WITHFILE:       -> None .
                ENDWI
*/

/* M58167A register offset for ASCUC */

#define CTTS_1      0x1L    /* Counter Ten Thousand of Seconds */
#define CHTS_1      0x3L    /* Counter Hundreds + Tenth of Sec */
#define CSEC_1      0x5L    /* Counter Seconds */
#define CMIN_1      0x7L    /* Counter Minutes */
#define CHRS_1      0x9L    /* Counter Hours */
#define CDOW_1      0xBL    /* Counter Day Of Week */
#define CDOM_1      0xDL    /* Counter Day of Month */
#define CMON_1      0xFL    /* Counter Month */
#define RTTS_1      0x11L   /* RAM - Ten Thousands of Seconds */
#define RHTS_1      0x13L   /* RAM - Hundreds + Tenth of Sec */
#define RSEC_1      0x15L   /* RAM - Seconds */
#define RMIN_1      0x17L   /* RAM - Minutes */
#define RHRS_1      0x19L   /* RAM - Hours */
#define RDOW_1      0x1BL   /* RAM - Day Of Week */
#define RDOM_1      0x1DL   /* RAM - Day Of Month */
#define RMON_1      0x1FL   /* RAM - Month */
#define ISS_1       0x21L   /* Interrupt Status Register */
#define ICR_1       0x23L   /* Interrupt Control Register */
#define CRES_1      0x25L   /* Counter Reset */
#define RRES_1      0x27L   /* RAM Reset */
#define STAT_1      0x29L   /* Status bit */
#define CDSTC_1     0x2BL   /* Go command */
#define SINT_1      0x2DL   /* Standby Interrupt */
#define TEST_1      0x3FL   /* Test mode */
```

Fig 4.2 : Suite

parmi un jeu d'utilitaires de communication permettant de télécharger et de lancer des logiciels (système d'exploitation ou autre) ou d'initialiser diverses cartes.

```
;Chargement des systèmes d'exploitation :

SYSLOAD PF1
SYSLOAD PF2
SYSLOAD CPU5

Initialisation du decoupage memoire :

INIMEM M
INIMEM MEV1
INIMEM MEV2

Chargement des logiciels : ( ACQ_FERA et ACQ_CAMAC sont les noms des fichiers
contenant les logiciels )

LOAD PF1 , ACQ_FERA
LOAD PF2 , ACQ_FERA
LOAD CPU5 , ACQ_CAMAC

Lancement des logiciels :

GO PF1 , ACQ_FERA
GO PF2 , ACQ_FERA
GO CPU5 , ACQ_CAMAC
```

Fig 4.3 : Exemple de fichier de lancement d'application .

4.2.2. Communication entre le système de développement et les unités centrales cibles

Le contrôle du dispositif d'acquisition s'effectuant à travers un système de développement, il est nécessaire de se munir d'utilitaires de communication entre ce dernier et les microprocesseurs cibles. Ces utilitaires permettent à un utilisateur connecté sur le système de développement de passer des commandes aux différentes unités centrales cibles et de recevoir les messages que celles-ci peuvent délivrer en retour. Ils permettent par ailleurs, le téléchargement dans les unités cibles (dépourvue de moyen de stockage durable d'information) des logiciels qu'elles devront exécuter.

Pour communiquer avec un microprocesseur cible, on peut utiliser en premier lieu, une liaison série RS232. Dans l'organisation de notre dispositif de test (Fig. 4.4), un utilisateur connecté sur le port k i ($< k < j$) du système de développement, peut en passant la commande :

TM p avec $m < p < n$

se trouver relié à l'unité centrale de son choix. Toute la communication s'effectue alors à travers le système de développement mais de façon

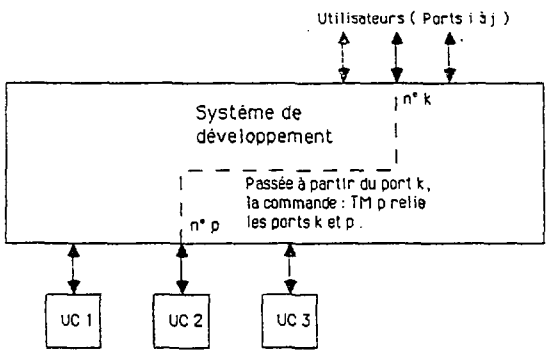


Fig 4.4 Usage de la commande TM de PDOS
Le mode transparent est symbolisé en pointillé.

transparente pour l'utilisateur qui a l'impression d'être directement connecté au microprocesseur cible. A titre d'exemple, on décrit ici deux utilitaires réalisés sur la base de cette commande.

TRANS <nom de l'UC cible> : relie l'utilisateur à un microprocesseur sur lequel il désire effectuer des opérations (passage de commande, recherche d'erreur, etc).

ONCDE <nom de l'UC cibles> , <commande> : Envoie la commande indiquée à l'unité centrale désignée sans que l'utilisateur ait à se connecter sur cette dernière. S'il y a un message en retour, il apparaît directement sur la console de l'utilisateur.

Ce mode de fonctionnement, tout à fait adapté pour des travaux en interactif avec les unités cibles, l'est beaucoup moins lorsqu'il s'agit de contrôler en ligne le déroulement de l'acquisition. En effet, il faut, pour recevoir les messages d'erreur éventuels émis par les différentes unités centrales cibles, prévoir une tâche en attente de caractères sur chacun des ports associés à ces dernières. Cette solution charge le système de développement en gestion de tâche et permet difficilement de centraliser les messages d'erreur sur un même terminal.

Il est donc intéressant d'envisager un mode de communication basé sur un dispositif de "boîte à lettre" [BON 87]. Entre l'unité centrale du système de développement et le microprocesseur cible, on place une mémoire qui sert d'intermédiaire pour la transmission. Pour communiquer des informations au microprocesseur, le système de développement commence par placer le message dans la mémoire puis il émet vers l'unité centrale cible, une demande d'interruption. Cette dernière active alors sur le système cible, un programme de lecture du message dans la mémoire. Les transmissions allant du microprocesseur cible vers le système de développement peuvent se baser sur le même principe de fonctionnement.

Ce type de communication est possible dans notre système puisque l'on dispose d'une liaison VMV entre les châssis VME, qui permet

au système de développement d'accéder à toutes les mémoires du dispositif et de transmettre des interruptions d'un bus VME à l'autre. Si une telle solution était mise en oeuvre, on n'aurait alors plus besoin, sur le système de développement que d'une seule tâche de contrôle qui inspecterait périodiquement les différentes boîtes à lettres et restituerait les messages qu'elles contiennent. Il reste malgré tout dans cette méthode, un problème à résoudre : celui des messages délivrés directement par les systèmes d'exploitations des unités cibles. Ceux-ci en effet, sont normalement émis vers un port de communication du microprocesseur et on doit trouver un procédé pour les orienter vers une mémoire. La solution retenue pour l'instant consiste à modifier les logiciels d'entrée/sortie ("BIOS") du système d'exploitation, de façon que les communications s'effectuent par l'intermédiaire d'une mémoire et non par une ligne série.

En ce qui concerne le téléchargement des logiciels dans les microprocesseurs cibles, ce sont sensiblement les mêmes problèmes et les mêmes solutions que précédemment qui apparaissent.

La commande DN (Download) de PDOS permet de transférer des fichiers de caractères par une voie série et nous a servi de base pour la réalisation de commande comme :

DNPGM <nom de l'UC cible>, <nom du programme>

qui télécharge le programme indiqué dans une RAM DISK associée à l'unité centrale désignée. La réalisation de téléchargement par voie série présente toutefois de gros inconvénients. Tout d'abord, on ne peut transférer sur une ligne RS232, que des caractères; il est donc nécessaire, pour télécharger un fichier de code exécutable, de transformer au préalable ce dernier en fichier de caractères puis de le convertir à nouveau à son format initial après transfert. Le temps total d'exécution de ces opérations ajouté à celui des transferts sur la ligne RS232, conduit à des téléchargements pouvant durer plusieurs dizaines de secondes, voire quelques minutes, ce qui représente une perte de temps importante en phase de test et de développement où l'on doit multiplier les essais.

Là encore, on peut résoudre le problème en utilisant les possibilités de communication offertes par la liaison VMV. Dans le système actuel de tests on dispose d'une RAM Disk située sur un châssis cible et utilisable à la fois par les unités centrales de ce châssis et par le système de développement (Fig. 4.5). Pour télécharger un fichier vers le châssis cible, il suffit alors de passer la commande TF (Transfer File) de PDOS avec les paramètres ci-dessous :

TF «nom du fichier» , «n° logique de la RAM DISK» . Le transfert s'effectue alors très rapidement et les microprocesseurs cibles peuvent exécuter le programme en l'appelant simplement par son nom.

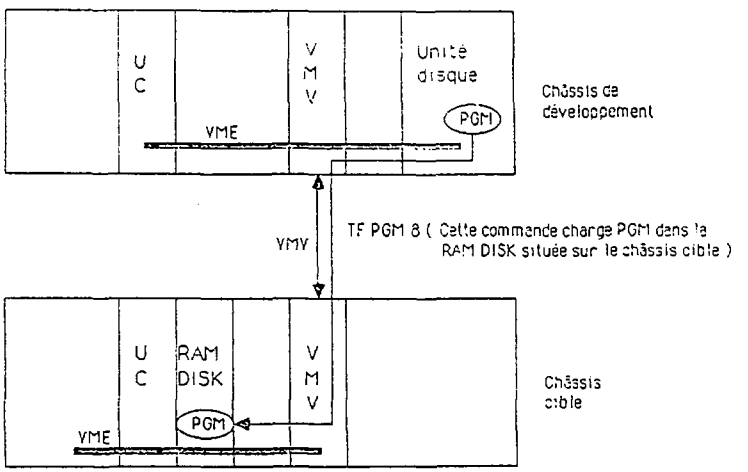


Fig 4.5 Utilisation d'une RAM DISK commune à deux châssis pour le téléchargement et l'exécution des logiciels

Toutefois, si pour une raison quelconque, on désire ne pas passer par l'intermédiaire d'une RAM DISK, on peut envisager comme précédemment un dispositif de téléchargement utilisant une boîte à lettres.

En effet, sur les cartes microprocesseur utilisées (CPU5, CPU20), la mémoire n'est pas accessible par le bus VME. On ne peut donc pas télécharger un fichier directement dans cette mémoire et il se révèle nécessaire de commencer par charger ce dernier dans une mémoire intermédiaire. Alors, par l'un des moyens de communications décrits ci-dessus, on active sur l'unité centrale cible, un utilitaire qui recopie le code à exécuter, dans la mémoire locale la commande suivante :

SYSLOGO <nom du fichier système> , <nom de l'UC cible>
permet de charger et de lancer le système d'exploitation voulu sur le microprocesseur désigné.

4.2.3. Logiciel de gestion de mémoire

4.2.3.1. Intérêt

On doit, sur l'ensemble du systèmpe d'acquisition, partager la mémoire entre les différentes unités centrales mises en oeuvre en tenant compte des fonctions à remplir : tampon de communication ou de téléchargement, zone de stockage d'événements ou de descripteurs etc. Ces zones de mémoire doivent être d'un accès aisé du point de vue de l'utilisateur, tout en présentant des protections contre les écrasements de données intempestifs. Il a donc été étudié un système de gestion de mémoire capable de fonctionner dans un contexte multiprocesseurs et fournissant des modes d'allocation de diverses complexités pouvant répondre à des besoins différents. Dans un souci de portabilité, on s'est attaché à rendre ce système indépendant de toute unité centrale et de tout système d'exploitation particulier.

4.2.3.2. Les découpages de la mémoire

Partant d'une mémoire physique, on effectue un premier découpage grâce à une fonction "définition" de "blocs primaires". Ces derniers qui constituent en quelque sorte les racines du système de gestion de mémoire, se composent d'un en-tête suivi d'une zone utile de travail dans laquelle on pourra réaliser des allocations de mémoires.

L'en-tête (Fig. 4.6) contient les paramètres de gestion de la zone utile. Afin de faciliter les accès au bloc, on lui attribue tout d'abord un nom, ce qui est plus pratique que de le désigner par une adresse et permet de ne pas se préoccuper de l'endroit où il se trouve en mémoire. Plusieurs processus (tâches) peuvent accéder à un bloc primaire, mais, pour éviter les erreurs, on autorise un seul d'entre eux à initialiser ou détruire le bloc. Le processus désigné est appelé propriétaire du bloc et a tous les droits d'accès sur ce dernier.

Nom et numéro du bloc
Nom du propriétaire
Adresses de début et de fin de la zone utile
Drapeaux de fonctionnement Droits d'accès / Mode d'allocation
Sémaphore et Compteur d'attachement
Gestion allocation avant : pointeur courant
Gestion allocation chaînée : pointeur sur la chaîne des zones libres pointeur sur la chaîne des zones occupées
Gestion allocation par bloc : taille d'une page en-tête gestion de champ de bit pointeur sur le premier répertoire pointeur sur le dernier répertoire

Fig 4.6 : Structure d'un en-tête de bloc.

Suivant le mode d'allocation demandé, on utilise l'un des trois systèmes de gestion inclus dans l'en-tête.

On donne ensuite les adresses de début et de fin de la zone utile. L'initialisation du bloc permet, par un positionnement adéquat de ses drapeaux de fonctionnement, de préciser les droits d'accès au bloc pour toutes les tâches autres que la tâche propriétaire, ainsi que le mode d'allocation de mémoire autorisé dans la zone utile . l'accès au bloc peut être réservé à son propriétaire, autorisé à toutes les tâches en lecture seule ou autorisé à toutes les tâches en lecture-écriture . D'autre part, on envisage trois modes d'allocation : allocation avant, allocation chaînée, allocation par blocs .

- Allocation avant : On définit un "pointeur courant" dans l'en-tête de bloc et on ne peut effectuer d'allocation qu'en avant de ce pointeur (fig 4.7.a) . C'est une méthode peu évoluée d'allocation qui ne permet de déallouer que la dernière zone allouée .

- Allocation chaînée : On alloue uniquement des zones contigues de mémoire et on crée un chaînage des zones libres et un chaînage des zones allouées (fig 4.7.b) . Pour faire une allocation, on prend, dans la chaîne des zones libres, une zone de taille requise (s'il en existe) et on la passe dans la chaîne des zones occupées . Inversement, à la déallocation, on passe une zone de la chaîne des occupées à la chaîne des libres . Ceci avec à chaque fois, les mises à jour adéquates dans les chaînages pour gérer les trous de façon optimale . Ce mode d'allocation ne sera pas implanté dans un premier temps .

- Allocation par blocs : Dans ce cas (fig 4.7.c,d),on alloue dans le bloc primaire, des blocs secondaires qui auront sensiblement la même structure que lui . Pour cela, on découpe le bloc primaire en pages dont la taille est fixée en fonction des besoins, puis on initialise, juste après l'en-tête, un champ de bits associant un bit à chaque page, ce qui permet de distinguer les pages libres (bit à 1) des pages occupées (bit à 0) . Un bloc secondaire sera alors un ensemble de pages contigues . Cependant, pour éviter à l'utilisateur de perdre de la place dans le bloc secondaire, on positionne son en-tête dans un répertoire situé juste après le champ de bits . Lorsque ce répertoire est plein, on en crée un autre que l'on chaîne avec le premier . Si on initialise le bloc secondaire en allocation par blocs, le champ de bits et le répertoire qui lui sont alors associés sont placés au début de sa zone utile .

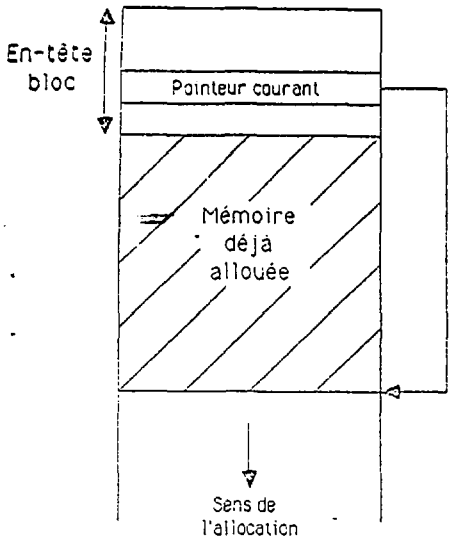


Fig 4.7 a : Allocation avant

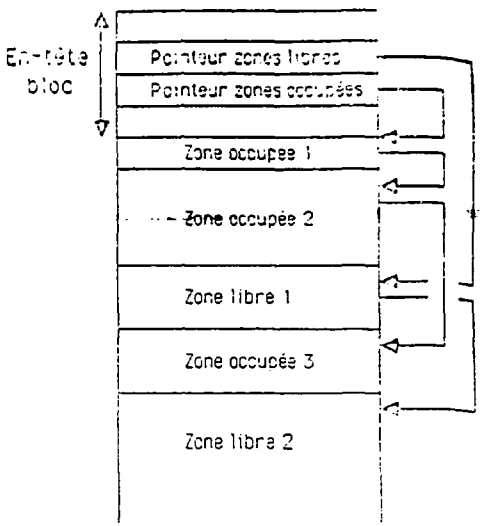


Fig 4.7 b : Allocation chaînée

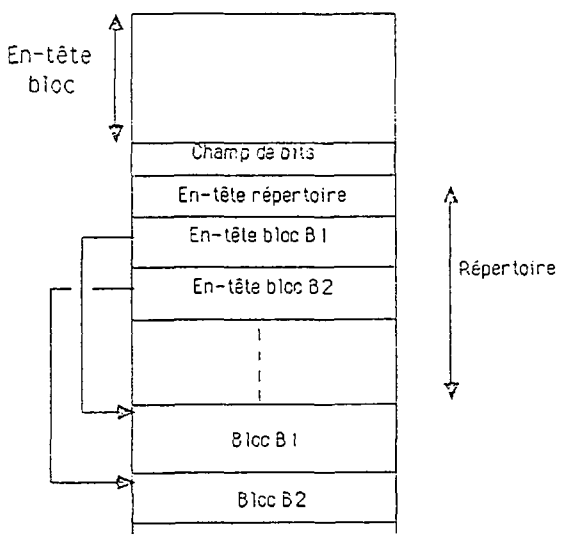
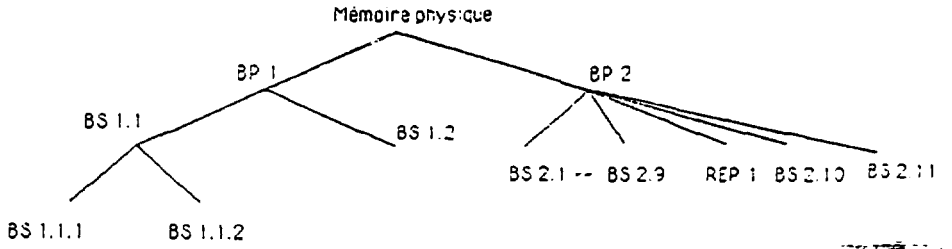


Fig 4.7.c : Allocation par blocs



La mémoire est découpée en deux blocs primaires : BP1 et BP2 .

On alloue dans BP1 deux blocs secondaires : BS1.1 et BS1.2 .
BS1.1 initialisé en mode d'allocation par bloc contient deux blocs secondaires : BS1.1.1 et BS1.1.2 . (On imbrique des allocations de blocs).

Dans BP2, on alloue 11 blocs secondaires B2.1 à B2.11 . En supposant que l'on dispose de 10 entrées dans le répertoire initial de BP2 (rep 0), on doit, pour référencer tous les blocs, créer un nouveau répertoire (rep 1).

Fig 4.7.d : Exemple d'allocations de mémoire

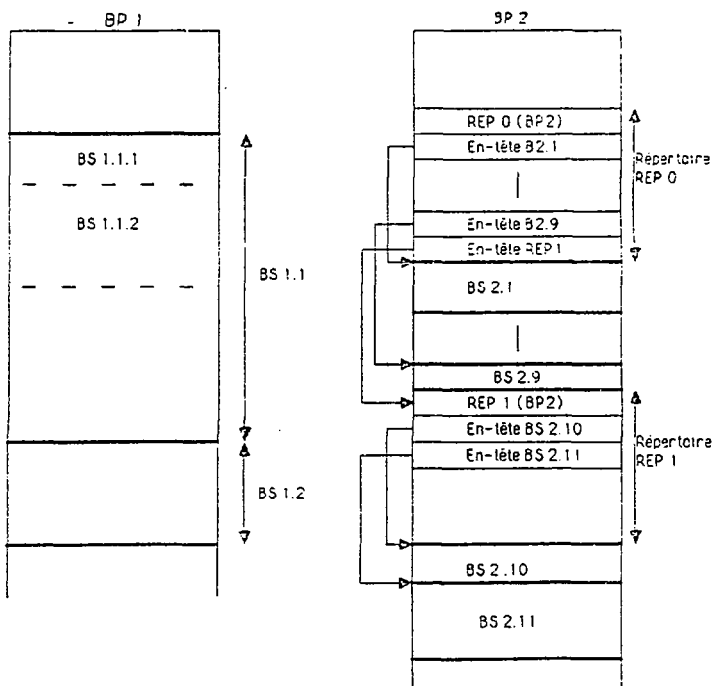


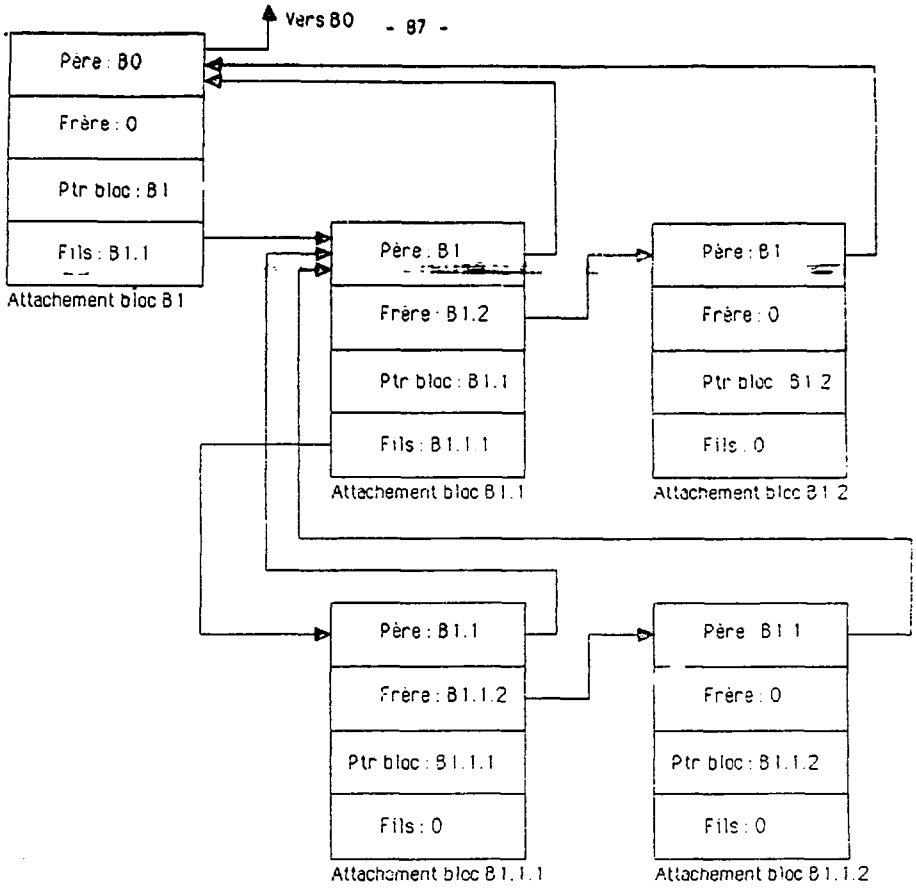
Fig 4.7.e : Découpage de la mémoire correspondant à l'exemple de la figure 4.7.d .

(Les blocs sont délimités par des traits gras)

Afin d'éviter les collisions lorsque plusieurs utilisateurs effectuent en même temps, des allocations de mémoire, on prévoit un sémaphore permettant de cadrer ces opérations dans des sections critiques . Par ailleurs, on dispose d'un compteur d'attachement indiquant le nombre de tâches utilisant le bloc à un instant donné . L'utilisation de ce compteur sera donnée dans le paragraphe 4.2.3.3 .

4.2.3.3. Mécanismes d'utilisation du logiciel

Un bloc reste inaccessible à tous les processus autres que son propriétaire tant que ce dernier ne l'a pas initialisé . Lorsque cette opération a été réalisée, on accède au bloc par l'intermédiaire d'une fonction attachement qui vérifie que les droits d'accès du processus demandeur sont compatibles avec les droits indiqués dans l'en-tête du bloc et retourne alors un pointeur sur ce dernier . Par ailleurs, elle incrémente le compteur d'attachement et met à jour un chaînage "historique" indiquant , pour chaque processus, les attachement qu'il a opéré (fig 4.8) . Ce chaînage permet, lors de l'attachement au bloc, de vérifier que l'on n'y est pas déjà attaché . Disposant du pointeur sur l'en-tête du bloc, on peut effectuer sur ce dernier, des lectures, des écritures, des allocations ou des déallocations de mémoire . Il faut noter que ce système de gestion permet de réaliser des allocations par blocs sur autant de niveaux d'imbrication que la taille de la mémoire le permet . Lorsqu'il n'a plus l'usage d'un bloc, un processus doit se détacher de celui-ci et de tous ses blocs fils . La fonction détachement utilise pour ce faire le chaînage "historique" défini précédemment . Cette fonction décrémente le compteur d'attachement . Il est aussi possible de détruire (déallouer) un bloc et de récupérer, pour un autre usage, la mémoire correspondante . Cette opération ne peut être réalisée que par le propriétaire du bloc et seulement lorsque le compteur d'attachement est nul (ce qui montre que plus personne n'utilise le bloc) .



Dans un bloc B0, on s'attache au bloc B1 .
Dans B1, on s'attache à B1.1 puis à B1.1.1 et B1.1.2 .
On s'attache ensuite toujours dans B1 au bloc B1.2 .

Fig 4.8 : Organisation du chaînage "historique" des attachements d'un processus aux différents blocs de mémoire

Avec un tel chaînage, on peut, partant d'un bloc, remonter à tous ses antécédents ou tous ses descendants auquels on est attaché.

4.2.4. Chronologie des opérations de démarrage d'une application

Afin de partir d'un état bien défini du système, on commence par effectuer une remise à zéro sur les différents bus VME cibles (fig 4.9). Cette opération peut être réalisée manuellement a l'aide de boutons poussoir ou par l'intermédiaire de la liaison VMV qui permet de transférer un ordre de remise à zéro à partir du système de développement vers les châssis cibles .

Il faut ensuite, sur chacun des microprocesseurs cibles, lancer l'exécution du système d'exploitation qui lui est associé dans le fichier de description de d'application . Le système d'exploitation sera, dans un premier temps, téléchargé par l'une des méthodes décrites au paragraphe 4.2.2 . On pourra ainsi réaliser des essais et générer un système d'exploitation optimisé pour les différentes unités centrales . A la fin de cette phase d'optimisation, on pourra implanter PDOS en mémoire morte (ROM) ou en mémoire permanente (mémoire vive + batterie), ce qui permettra d'éviter les opérations de téléchargement . L'exécution du système d'exploitation peut être lancée, soit directement lors de la remise à zéro des unités centrales cibles, soit en passant à ces dernières, par une des méthodes décrites au paragraphe 4.2.2, une commande appropriée .

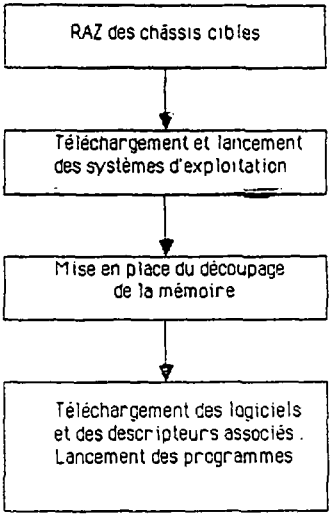


Fig 4.9 : Chronologie des opérations de lancement de l'application .

L'étape suivante consiste à mettre en place, les éléments de base du système de gestion mémoire . Le fichier de description de l'application spécifie les blocs à définir ou à allouer et la façon dont on doit les initialiser . A partir de ces informations, l'unité centrale du système de développement peut réaliser les opérations correspondantes et ceci, directement grâce à la liaison VMV qui lui permet d'accéder à toutes les mémoires du dispositif d'acquisition . Si cette liaison n'existait pas, la mise en place de la gestion de mémoire devrait s'effectuer en passant par l'intermédiaire des unités centrales cibles indiquées dans les chemins d'accès aux différentes mémoires . On doit ensuite, de la même façon, initialiser les différentes RAM DISK qui seront utilisées pour le téléchargement des logiciels .

Ces derniers, composés des programmes à exécuter ainsi que des descipteurs qui leurs sont nécessaires pour fonctionner, peuvent alors être téléchargés dans les mémoires associées aux unités centrales cibles et lancés en envoyant à ces dernières, les commandes appropriées.

4.3. Synchronisation des processeurs de filtrage

4.3.1. Présentation générale du problème

Sur le bus VME2, plusieurs unités centrales (PF) fonctionnent en parallèle pour l'acquisition et le filtrage des évènements . L'algorithme présenté ici, permet de déterminer, pour chaque nouvel événement incident, quel microprocesseur devra le lire et le traiter . Afin de profiter au maximum des avantages du parallèlisme, le temps d'exécution de l'algorithme devra être court devant celui des traitements et, pour ce faire, les fonctionnalités matérielles offertes par le VME seront largement utilisées .

C'est dans ce souci de rapidité que l'on ne prévoit pas de maître (au sens logiciel) des processeurs PF . En effet, la gestion par échange de questions et réponses impliquée alors, représenterait une perte de temps beaucoup plus importante que dans le cas où les processeurs se synchronisent sans contrôle extérieur, en exécutant, de façon asynchrone les uns par rapport aux autres et en tâche de fond, un

algorithme approprié .

Avec une telle organisation, ce dernier devient, dans son principe, totalement indépendant du nombre d'unités centrales mises en oeuvre .

Nous désignerons par la suite le processeur "courant" comme étant celui qui est en cours de lecture d'un événement dans les cartes IFV et le processeur "suivant" comme étant l'unité centrale retenue pour l'acquisition de l'événement à venir . L'algorithme de synchronisation doit assumer les trois fonctions suivantes :

- Déterminer le processeur suivant : Pendant ou en fin de lecture d'un événement, on détermine quel microprocesseur sera le suivant.

- Basculer du processeur courant sur le suivant : Après l'arbitrage décrit ci-dessus, on doit indiquer au processeur retenu qu'il est le suivant et lui fournir les pouvoirs nécessaires pour l'acquisition du prochain événement .

- Détecter la présence d'un événement : Lorsqu'un processeur a été désigné comme suivant, il faut l'avertir, le cas échéant, de la présence d'un événement dans les files d'attente d'entrée . C'est ce signal qui provoque la transition de l'état suivant à l'état courant du processeur et donc, déclenche la lecture de l'événement .

Nous allons maintenant présenter deux algorithmes de synchronisation possibles, l'un basé exclusivement sur des mécanismes matériels et l'autre alliant matériel et logiciel .

4.3.2. Synchronisation par arbitrage du bus VME

Une solution facile à mettre en oeuvre et très efficace en rapidité d'exécution consiste à utiliser exclusivement les fonctionnalités matérielles définies dans la norme VME . Cette dernière prévoit pour un maître, plusieurs modes de libération du bus parmi lesquels :

- ROR : "Release On Request". Le maître libère le bus dès qu'une autre demande ("Bus Request") intervient .

-RBCLR : "Release on Bus Clear". Le maître ne libère le bus que s'il reçoit de la part de l'arbitre du bus, un signal "Bus Clear"

signifiant qu'une demande plus prioritaire est pendante . Toutes les unités centrales sont placées sur un même niveau de priorité pour les demandes d'accès au bus et exécutent en tâche de fond, l'algorithme présenté fig. 4.10 .

Le processeur qui, le premier, prend le contrôle du bus, entre dans une boucle d'attente d'événement issu des cartes IFV . Lorsqu'un événement se présente, le processeur le stocke, pour traitement, dans sa mémoire locale et passe alors en mode ROR . Les autres unités centrales, ne pouvant jusque là, accéder au bus, ont maintenu leurs demandes d'accès respectives et un arbitrage a lieu (au sens VME) permettant de désigner le processeur qui effectuera l'acquisition de l'événement à venir. Toutes les unités centrales étant connectées à une même ligne de demande de bus, leur priorité d'accès est déterminé par leur position géographique dans le châssis ; ceci du fait de l'organisation en "daisy-chain" des lignes "Bus Grant" d'attribution du bus à un maitre.

Cette façon de procéder est très optimisée du point de vue temps d'exécution car d'une part, la détermination du suivant et le basculement courant/suivant s'effectuent très rapidement dans une même opération d'arbitrage du bus VME et, d'autre part, la boucle de test utilisée pour la détection d'événement à lire conduit à des temps de réponse très courts par rapport au cas où cette fonction serait assurée par une demande d'interruption .

Cependant, cette méthode ne permet pas qu'un processus puisse effectuer un quelconque traitement pendant l'attente d'un événement . De plus, elle présente un inconvénient majeur : le bus VME2 ne peut servir qu'à la gestion de la synchronisation des processeurs PF et à la lecture des événements . Il ne peut, en particulier, être utilisé pour réaliser un système de communication par boîte à lettre . En effet, considérons l'exemple de la figure 4.11 . Trois processeurs PF sont en service et la mémoire MEM sert de boîte à lettre pour la communication entre ces derniers et le système de développement . Supposons que PF2 ait, à un instant donné, le contrôle du bus VME en mode RBCLR pour la lecture d'un événement . Aucun des autres PFi ne peut alors accéder au bus . Par contre, l'unité centrale du système de développement peut, par l'intermédiaire de la liaison VMV, demander un accès prioritaire au bus

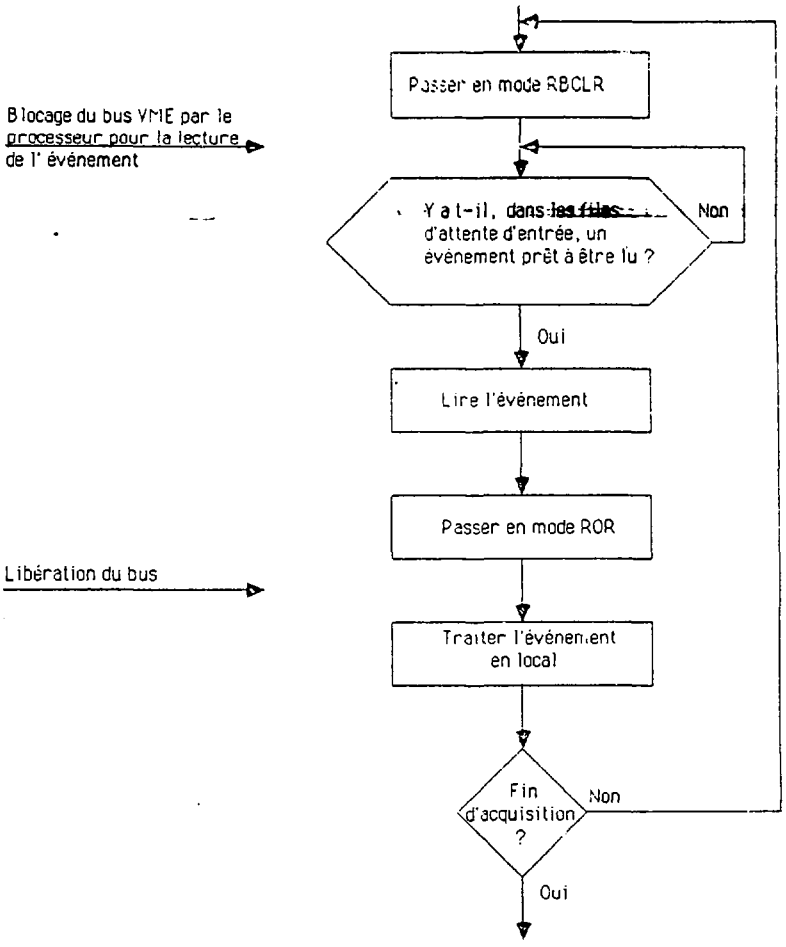
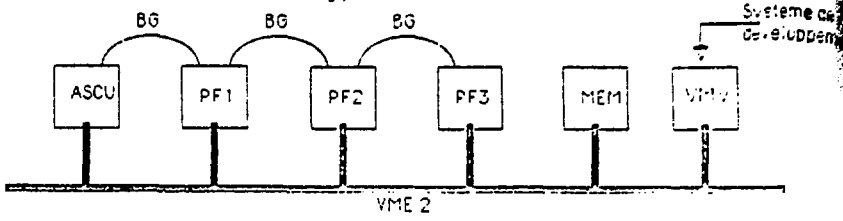


Fig 4.10 : Synchronisation des processeurs PF par arbitrage du bus VME.
Chaque processeur PFI éxécute en tâche de fond, l'algorithme ci-dessus.



ASCU : Carte contrôleur de bus VME .
PF1 : Processeur de lecture et de filtrage des événements.
MEM : Mémoire boîte à lettres pour des utilitaires de communication .
BG : Ligne "Bus Grant" d'attribution du bus à un maître .

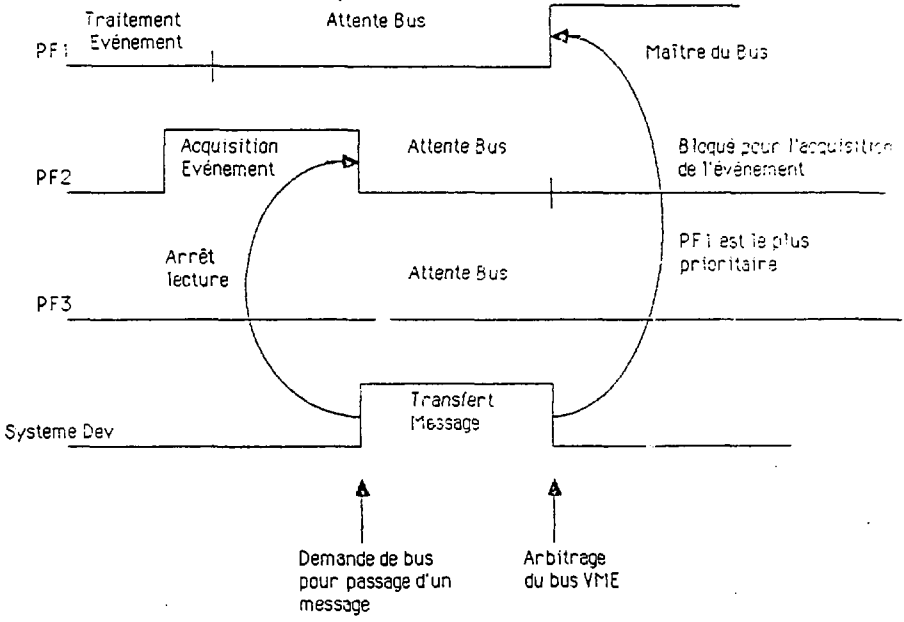


Fig 4.11 : Exemple de problème pouvant survenir avec une synchronisation des processeurs n'utilisant que des fonctionnalités matérielles.
Les crénaux représentent à l'état haut : un état maître du bus.

pour passer un message dans la mémoire boîte à lettre . Après dépôt du message, cette unité centrale libère le bus qui fait alors l'objet d'un arbitrage pour déterminer quel proceseur en sera le maître . Si PF1 est prêt pour l'acquisition d'un événement, c'est lui qui obtient le contrôle du bus alors que PF2 n'a pas terminé l'acquisition de l'événement qu'il avait commencé à lire. On a donc un problème de synchronisation et seul un processus logiciel permettrait de faire en sorte que PF1 libère le bus au profit de PF2 .

Nous nous sommes donc orientés vers un algorithme utilisant à la fois des éléments matériels et logiciels .

4.3.3. Algorithme de synchronisation retenu

Cet algorithme s'articule autour des quatre éléments logiciels suivants :

- drapeau S : S = 1 indique qu'un processeur suivant a été déterminé .

S = 0 indique qu'aucune unité centrale n'est encore prête à lire l'événement à venir .

- drapeau C : C = 1 indique qu'il existe un processeur courant (en cours de lecture d'un événement)

C = 0 indique qu'aucun processeur n'est en cours d'acquisition d'événement .

- Une pile LIFO (Last In First Out) dans laquelle peuvent venir se consigner des processeurs en attente d'événement . Par logiciel, une pile est plus facile à gérer qu'une file, c'est pourquoi nous avons pris cette orientation .

- Un sémaphore permettant de protéger l'accès aux trois éléments précédents par des sections critiques .

Ces variables seront placées en mémoire globale accessible de tous les processeurs PF .

Avant de détailler l'algorithme, nous devons préciser deux points matériels . Le bus VME2 est muni d'une carte contrôleur (Advanced System Control Unit de FORCE COMPUTERS) supportant un arbitre de bus que l'on configure en mode PRI (voir annexe 2) . Les quatre lignes (BR0 à BR3) de demande de bus sont donc hiérarchisées et leurs

priorités respectives sont fixées . Les unités centrales sont toutes connectées à une même ligne de demande de bus, elles ont donc de ce point de vue, la même priorité . Par ailleurs, les différentes files d'attente en entrée (cartes IFV et M) étant synchronisées, on désigne une des cartes IFV comme "porte-parole" pour signaler la présence d'un événement à lire . Cette carte peut, en particulier, délivrer un signal, qui, transitant par la carte ASCU, intervient sur le bus VME, sous la forme d'une interruption dont le niveau est programmable . Or, Chacune des cartes PF est configurée de façon à ne prendre en compte, qu'un seul niveau d'interruption. On peut donc activer le processeur "suivant" par une interruption issue de IFV en programmant correctement le niveau de celle-ci .

Nous pouvons maintenant décrire comment sont traités les trois points de l'algorithme de synchronisation .

La détermination du processeur suivant intervient dès le début de l'organigramme (fig4.12.a et b) . Un test du drapeau S permet de savoir si un suivant a déjà été sélectionné . Si oui, le processeur PFi se positionne dans la pile d'attente . Sinon, il teste le drapeau C et, le cas échéant, la présence d'un événement dans les tampons d'entrée et termine, soit dans l'état suivant , soit directement dans l'état courant suivant le résultat des tests . Etre dans l'état suivant consiste pour un processeur PF à se mettre en attente d'une interruption en provenance de IFV . Cette interruption peut être autorisée par le processeur lui même ou par le processeur courant en fin de lecture d'un événement . A ce moment, en effet, le processeur courant teste lui aussi, si un suivant a été déterminé . Si oui, il positionne les registres de la carte contrôleur ASCU de façon que la prochaine interruption fournie par la carte IFV parvienne au processeur désigné comme suivant . Cette opération d'ailleurs, participe au basculement du processeur courant sur le processeur suivant . Sinon, si la pile n'est pas vide, il désigne le premier élément de celle-ci comme suivant en initialisant comme ci-dessus les registres de la certs contrôleur . Si la pile est vide et qu'aucun suivant n'est ou ne s'est désigné, alors, tous les processeurs PF sont en cours de traitement d'un événement . Il faut dans ce cas, attendre qu'au moins un d'entre eux ait terminé et reprenne l'exécution de l'algorithme à son début .

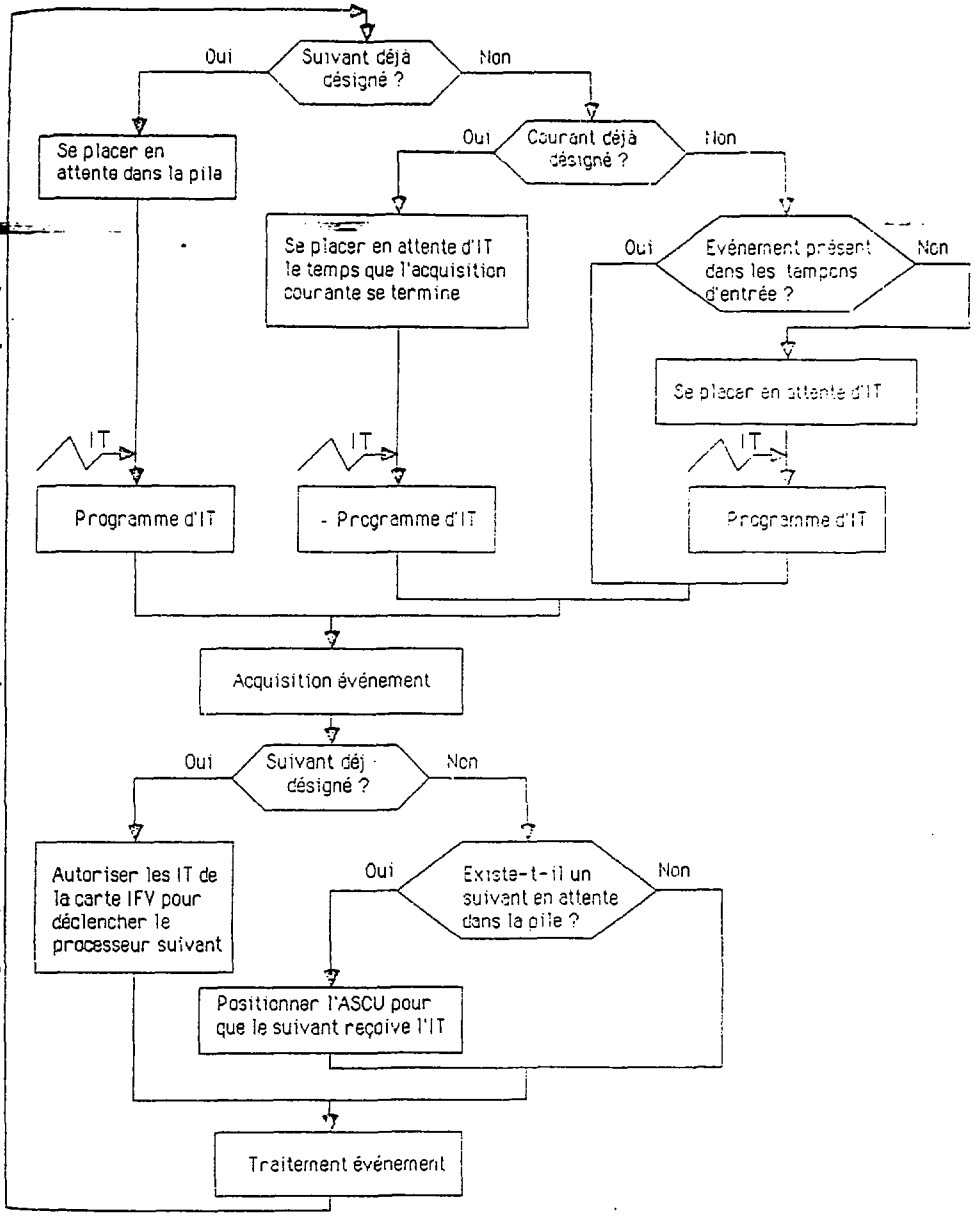


Fig 4.12.a : Principe de l'algorithme de synchronisation des processeurs PF

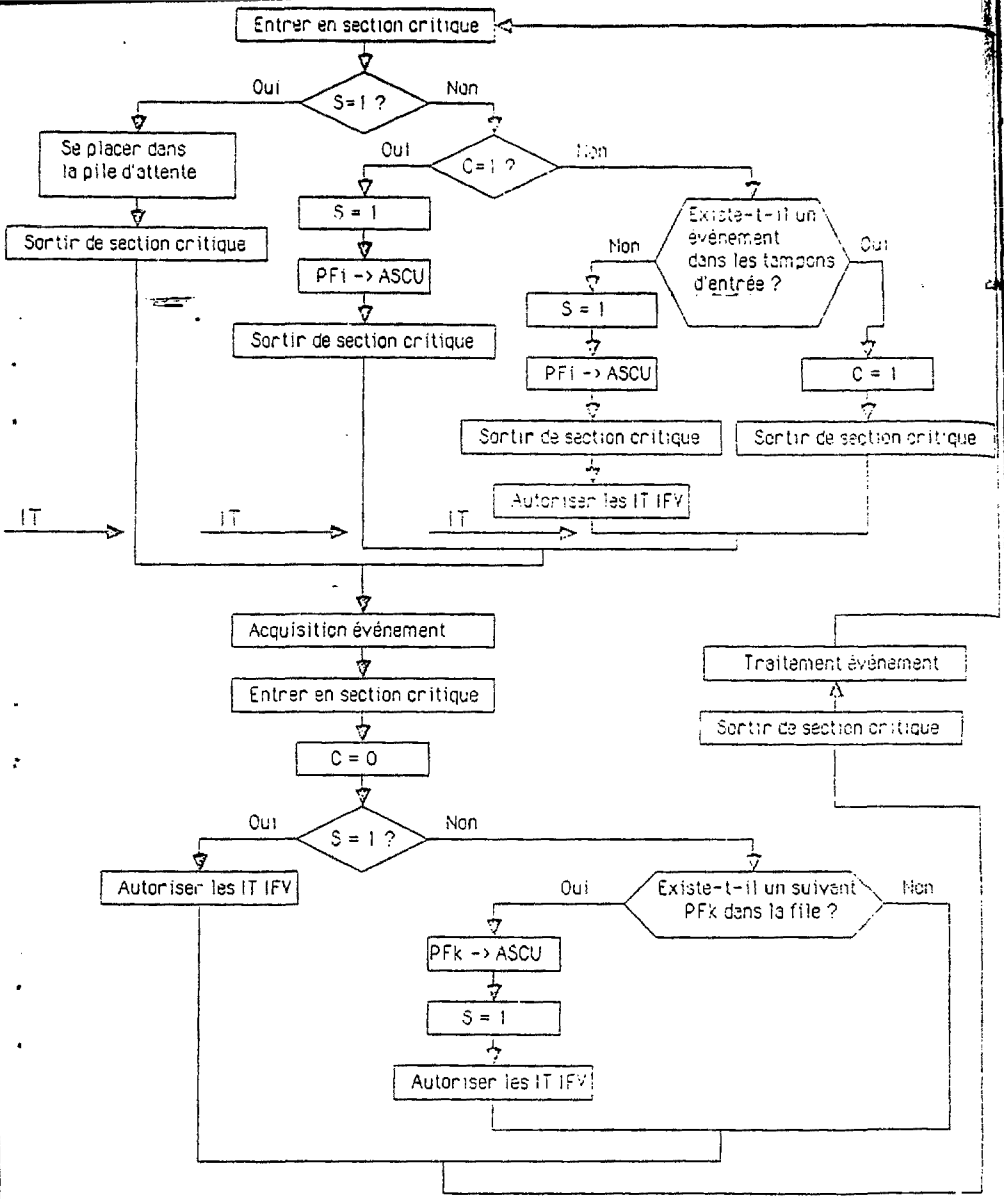


Fig 4.12.b : Algorithme de synchronisation des processeurs PF.

PFk->ASCU signifie : positionner les registres de l'ASCU de façon que la prochaine interruption émise par la carte IFV parvienne à PFk.

Le basculement du processeur courant sur le suivant s'effectue en positionnant le drapeau S à 1 et en initialisant les registres de la carte contrôleur de façon que la prochaine interruption fournie par la carte IFV parvienne au processeur désigné comme suivant .

Pour détecter la présence d'un événement, deux méthodes sont possibles : recevoir une interruption ou tester un bit "file vide" sur la carte IFV "porte-parole" . Pour mettre un processus en attente d'interruption, on place la tâche qui exécute l'algorithme en attente d'événement PDOS . Le programme d'interruption positionne cet événement, ce qui réveille la tâche et déclenche la lecture des données . Ce n'est que dans le cas où PFi accède directement à l'état courant (en partant du début de l'organigramme), qu'il teste par logiciel la présence d'un événement dans les cartes tampon . Ce cas, qui suppose la réunion de plusieurs conditions sera peu fréquemment rencontré et le plus souvent, c'est une interruption qui déclenchera la lecture d'un événement . Ceci conduit à des temps de réponse plus longs que ceux que l'on obtiendrait avec une boucle d'attente mais l'algorithme reste ici plus ouvert car on peut envisager pendant que la tâche d'acquisition est en sommeil (attente d'interruption), de réveiller une tâche de traitement échantillonné ou de communication par exemple .

De plus, pour limiter le nombre d'arbitrages du bus VME, on place les processeurs en mode RBCLR pour l'exécution des sections critiques . En effet, il n'est pas nécessaire d'autoriser les autres processeurs à tenter d'accéder en section critique alors que l'on s'y trouve déjà . Il faut noter que ceci ne bloque nullement le bus car la durée d'exécution des sections critiques reste courte devant les temps de lecture et de traitement d'un événement .

CONCLUSION

Les différents éléments du système d'acquisition sont, à l'heure actuelle, presque tous réalisés ou en cours de dévelopement. Les codeurs de charge et de temps ont étés reçus et testés, le codeur d'amplitude est maintenant opérationnel et une pré-série de 10 éléments est en cours de montage. Si les cartes IFV et DC sont en cours de test, la carte RS reste, elle, encore à l'étude. Cependant, on peut envisager pour les premières expériences, de la simuler par logiciel même si cela doit altérer provisoirement les performances du système d'acquisition. Les élément constitutifs du dispositif VME sont réunis et on compte effectuer les premiers essais avec deux ou trois processeurs PF. Toutefois, on peut ici se poser la question de l'évolution du système. En effet, lorsque nous avons acheté nos cartes, l'extension standard du bus VME était le VMX; mais, la firme MOTOROLA ayant étudié une extension baptisée VMX32 et faisant valoir celle-ci, une commission de standardisation a synthétisé les deux propositions dans la norme VSB qui sera donc en vigueur à l'avenir. Nous serons donc amenés, à moyen ou long terme, à utiliser des cartes au standard VSB mais ceci pourra se faire de façon progressive car il suffira à chaque fois de changer un ensemble (PFi + MEVi) ou (CPU5 + M). On pourra donc mettre le système à jour facilement et sans avoir à investir en une seule fois, le prix du changement de tout le dispositif. La carte interface avec le calculateur MODCOMP a été réalisée et doit être testée. Du point de vue matériel, rien n'a encore été établi quant au système de visualisation et de contrôle. Par contre, un logiciel de visualisation est en cours de développement dans le cadre d'une collaboration entre les laboratoires de l'Institut National de la Physique Nucléaire et de la Physique des Particules (IN2P3) et c'est ce logiciel que nous utiliserons. Par ailleurs, si les logiciels de fonctionnement du système sont définis (téléchargement, synchronisation des processeurs etc), il n'en est pas de même pour les fonctions d'utilisation de ce dernier. Il s'agira donc de mettre en place, un système permettant de réaliser l'interface entre des commandes "utilisateur" et les utilitaires déjà définis. Nous n'avons pu effectuer de tests de vitesse sur l'algorithme de

synchronisation des processeurs, ceci en raison des points "périphériques" (communication avec les unités centrales cibles, gestion de mémoire etc) qu'il nous a fallu étudier. Certains points peuvent cependant déjà être notés. L'architecture retenue pour le système d'acquisition, modulaire et structurée, rend ce dernier très ouvert à des perfectionnements ultérieurs et lui permet de continuer à fonctionner (avec des performances réduites toutefois) même si certains éléments tombent en panne. De plus, on peut utiliser un même type de système pour réaliser l'acquisition d'expériences autres que les expériences multidétecteurs. De la même façon, certains points logiciels (système de description d'application, système de gestion de mémoire, utilitaires de communication pour les aplications basées sur le système d'exploitatoin PDOS) pourront être employés dans le cadre d'autres applications VME.

REFERENCES BIBLIOGRAPHIQUES

[BON 87] La communication UNIX - PSOS pour les applications temps réel sur bus VME : J.M. Bonnaud, J.L. Scémia.
Mini et Micros n° 275-276.

[BOU 87] "DELF", a large solid angle detection system for heavy fragments : R. Bougault, J. Duchon, J.M. Gautier, A. Genoux-Lubain, C. Lebrun, J.F. Lecolley, A. Lefebvres, M. Louvel, P. Mosrin, R. Regimbart.

[CAR 86] PDOS - The realtime operating system solution for machine control : Reed W. Cardoza.
Softtalk and Hardfacts, vol. 4, N° 3, Sept. 86.

[CES] 70 route du Pont-Butin, case postale 122
1213 PETIT-LANCY.

[CHA 86] Etude de l'ensemble Groupement-Transfert : F. Chanu, M. Margerie.
Projet de fin d'études.

[DRO 85] Mise en oeuvre d'un multidétecteur de noyaux légers : A. Drouet.

[ESO 83] CAMAC Updated Specifications. Comité ESONE.
EUR 8500 EN, Vol. 1 et 2.

[EYRI] EYRING Reseach Institute Inc
1450 West 820 North, PROVO, UTAH 84601

[FORC] FORCE COMPUTERS GMBH
European Headquarters
Daimlerstraße 9.D.8012 OTTOBRUNN

[GAV 85] Trends in new collider experiment data acquisition systems : Ph. Gavillet .
NIM A235

[GUE 84] Programmation du microprocesseur CAB pour optimisation de l'acquisition des données dans une expérience de physique nucléaire : G. Guérin .
Rapport de fin d'étude .

[KER 84] Le langage C : B.W. Kernighan et D. Ritchie .

[LEC 85] Results of the second level software trigger used in the NA10 experiment at CERN : J. Lecocq, J.J. Blaising, J.P. Froberger, Ph. Klein, J.M. Meyer .
NIM A237 .

[LECR] LECROY SA
European Headquarters
Route du Nant d'avril 101
ch-1217 MEYRIN 1-GENEVE SUISSE

[MAR 85] Le configurateur et ses modules .
Manuel descriptif des modules et des montages : R. Margaria, J. Tillier .

[MOTO] MOTOROLA Semi Conductor Products Inc
Po Box 20912 PHOENIX ARIZONA 85036

[POS 85] Etude d'une chaîne d'acquisition pour les grands détecteurs au GANIL : H. Postec .
rapport DEA RS 85-09 .

[RIC 85] Acquisition de données sur les grands détecteurs au GANIL.
Codage : A. Richard .
Rapport interne AR AH 16/4/85 .

[TIL 83] Adaptation du microprocesseur CAB à la chaîne d'acquisition du GANIL : J. Tillier .

[TRI 87] Etude et réalisation de codeurs d'amplitude : M. Tripon . RS 87-01

[VAI 83] GANIL Acquisition System : D. Vaillant, P. Bertrand, J. Clément, B. Raine, J. Tillier, R. De Tourreil . IEEE Transactions on Nuclear Science Vol NS30 N°5 oct 83

[VIL 84] Gestion de données du "Château de Cristal" : M.M. Vilard . Compte-rendu réunion du 1-2 Fev 84 .

BIBLIOGRAPHIE

[BIZ 86] A plastic multidetector for light nuclei identification at GANIL : G. Bizard, A. Drouet, F. Lefebvres, J.P. Patry, B. Tamain, F. Guilbault, C. Lebrun .
NIM A244 .

[KNO 79] Radiation detection and measurement : G.F. Knoll .

[MAT 83] Le CAB : P Matricon .
Forum sur la microinfrmatique en physique nucléaire et en physique des particules 19-21 sept 83

[POC 87] Two particles correlations at small relative momenta for ^{40}Ar induced reactions on ^{197}Au at E/A = 60 Mev : J. Pochodzalla, C.K. Gelbke, W.G. Lynch, M. Maier, D. Ardouin, H. Delagrange, H. Doubre, C. Grégoire, A. Kyanowski, W. Mittig, A. Peghaire, J. Peter, F. St Laurent, B. Zwieglinski, G. Bizard, F. Lefebvres, B. Tamain, J. Quebert, Y.P. Viyogi, W.A.Friedman, D.H. Boal .
Physical Review C Vol 35, Num 5 may 87

[SAM 68] Instrumentation électronique en physique nucléaire . (mesure de temps et d'énergie) : J.J. Samueli, J Pigneret, A. Sarazin .

[VOLa 80] The length oh input data buffers in real time data acquisition systems : P. Volkov
NIM A171 .

[VOLb 80] Data buffering between a microprocessor and a minicomputer in a data acquisition system : P. Volkov .
NIM A171.

ANNEXE 1

COMPTE-RENDU DES TESTS DU SYSTEME FERA

Hervé Postec

GANIL Caen Le 14/3/86

COMPTE RENDU DES TESTS DU SYSTEME FERA

Ce compte rendu présente les résultats de tests effectués sur un ensemble 4300 de LECROY, visant à évaluer la linéarité différentielle des codeurs et les performances du bus FERA.

1/ Linéarité différentielle.

11/ Définition utilisée.

On présente à l'entrée du codeur, des valeurs variant aléatoirement entre 0 et la pleine échelle, puis on trace le spectre résultant : en abscisse, les canaux, en ordonnée, le nombre de fois que chacun d'entre eux est touché. Pour un codeur parfait, ce spectre serait une droite horizontale. En fait on observe toujours une certaine dispersion et on évalue la linéarité différentielle comme étant : LD = (max du spectre-moy du spectre)/ moyenne. Cette méthode statistique est moins exacte que si l'on considérait les différences des canaux successifs pris deux à deux, mais on constate qu'elle donne des résultats identiques.

12/ Test des QDC.

* Montage.

On assemble un dispositif permettant de "balayer" toute la gamme d'entrée du codeur, de façon aléatoire. Pour cela, deux générateurs sont utilisés comme indiqué sur le schéma 1/1. Le générateur A délivre la GATE au codeur et déclenche le générateur B. Ce dernier, asservi en amplitude par un générateur de rampe, fournit à l'entrée analogique du codeur, un signal carré modulé en amplitude. On synchronise A et B de façon à obtenir le diagramme de temps 1/2.

A et le générateur de rampe étant totalement découplés en phase, on intègre le signal analogique pour des amplitudes variant aléatoirement entre 0 et le maximum délivré par le générateur de rampe. En réglant cette amplitude, on peut balayer la pleine échelle du codeur.

* Résultats obtenus.

On obtient par cette méthode une linéarité différentielle d'environ +/- 15% (cf spectre 1).

Une loupe effectuée sur le spectre montre une périodicité de ce dernier sur 8 canaux qui peut signifier des problèmes de codage (cf spectre 2).

De plus, on note sur les faibles canaux du spectre, une oscillation qui reste à expliquer (cf spectre 3).

Les spectres 4 montrent à conditions d'entrée égales, la médiocrité des performances du codeur 4300 devant celles d'un module QDC 2249W de LECROY.

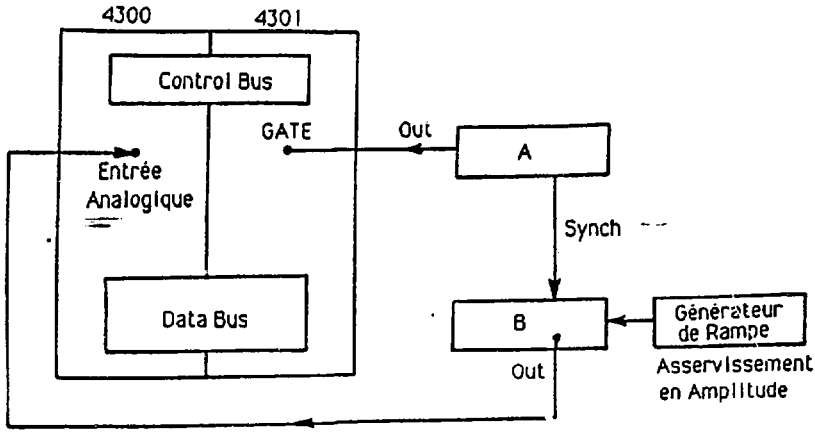


Schéma 1/1 : Elaboration de spectres.

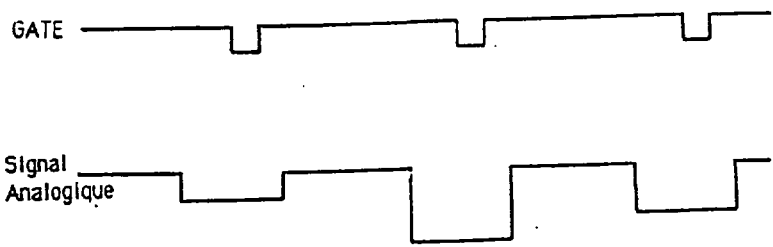
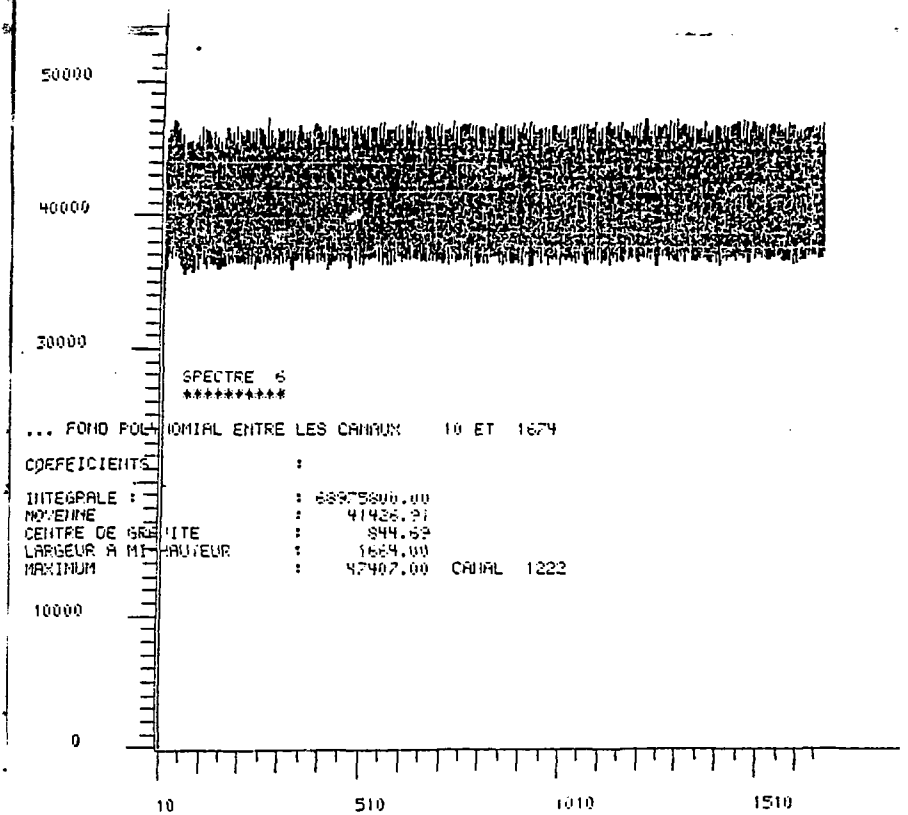


Schéma 1/2 : Synchronisation des signaux.



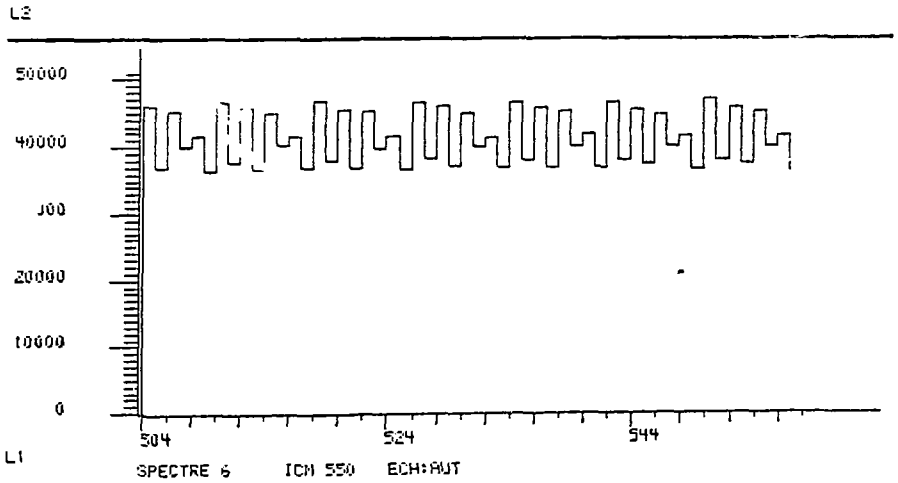
1 Zoom sur la partie linéaire du spectre.

Linéarité différentielle ± 1 %

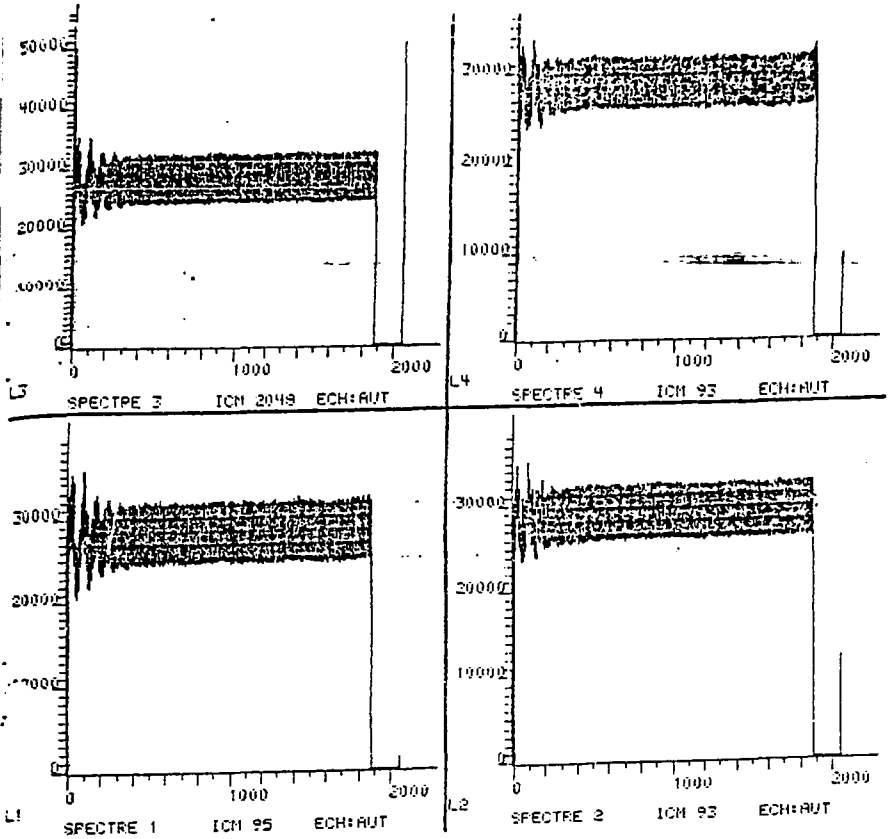
SPECTRE 6
****

... FOND POLYNOMIAL ENTRE LES CANAUX 504 ET 552

COEFFICIENTS :
0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00 0.0000E+00
INTEGRALE : : 2331471.00
MOYENNE : 41323.53
CENTRE DE GRAVITE : 530.44
LARGEUR A MI-HAUTEUR : 53.00
MAXIMUM : 46925.00 CANAL 550

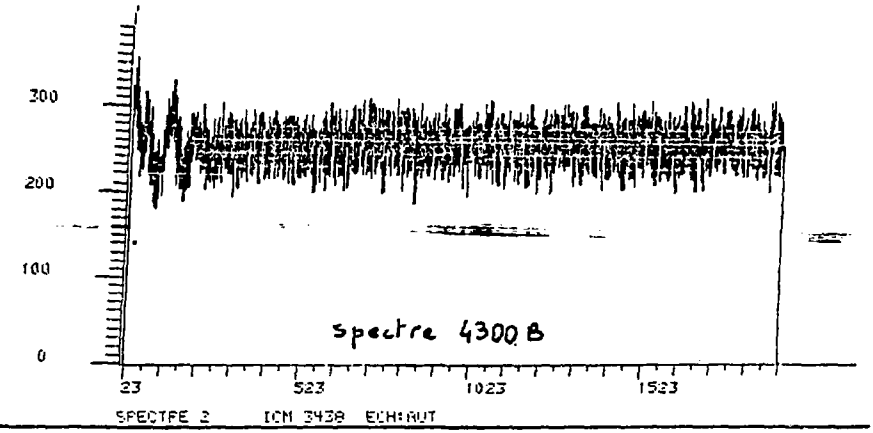


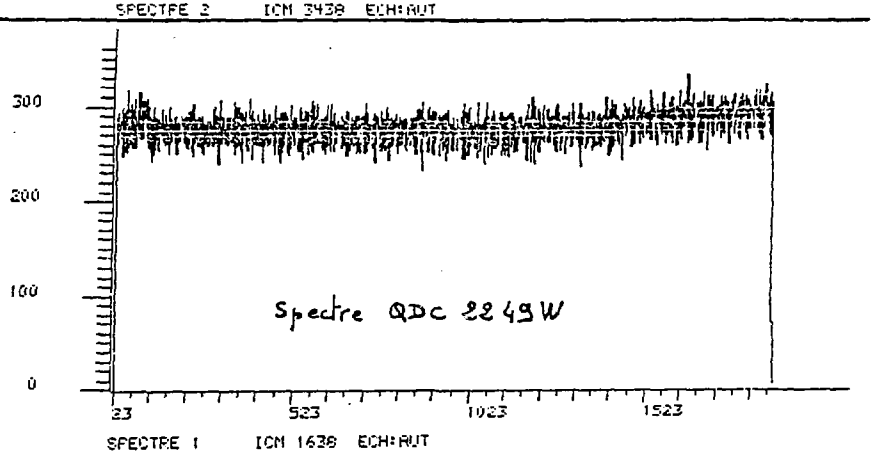
2 Périodicité du Spectre



3 Comparaison de 4 voies du codeur

4300 FRAN





4 Comparaison du codeur FERA
avec un autre QDC

2/ Test du bus FERA.

Ces tests avaient pour but de vérifier les caractéristiques données dans la documentation et d'étudier son mode d'utilisation.

2.1 Test avec un seul codeur sur le bus.

* Un schéma du montage est donné schéma 2/1.

* Résultats.

On laisse les entrées analogiques du codeur "en l'air", car on ne s'intéresse ici qu'au mode de transmission des données et non à leur valeur. La visualisation des signaux : GATE, ROO, WSO permet de mesurer la vitesse de transfert des informations sur le bus. Le chronogramme obtenu est donné schéma 2/2.

->1 = Temps entre le début du codage et le moment où le codeur demande l'accès au bus pour émettre ses données. La documentation donne les temps de conversions
4300 10 bits : 4,8 microsecondes.
4300 11 bits : 8,5 microsecondes.
Les mesures donnent pour le temps n°1 les valeurs suivantes :

	Sans suppression de 0	Avec suppression de 0
4300 10 bits	5,3 microsecondes	7,6 microsecondes
4300 11 bits	9,3 microsecondes	11,5 microsecondes

Ce tableau montre que : pour les cas où l'on n'a pas de suppression de 0, les données ne sont disponibles en sortie que 0,5 à 1 microseconde après la fin du temps de codage indiqué par LECROY. Or, pour l'utilisateur, c'est le temps au bout duquel les données sont accessibles qui importe, donc pour un codeur 11 bits il faudra compter 9,3 microsecondes de temps de codage sans suppression de 0 et 11,5 dans le cas de suppression de 0. Il faut noter que la suppression de 0, s'effectue en 2,3 microsecondes au lieu des 1,6 annoncées dans la documentation.

->2 = Donné à titre indicatif. On peut le considérer comme le temps de réaction du 4301 au signal de requête de bus émis par le codeur.
4300 10 bits : 50 ns.
4300 11 bits : 60 ns.

->3+4 = C'est le temps qui s'écoule entre l'émission de deux données successives. C'est lui qui conditionne la vitesse instantanée d'acquisition sur le bus FERA. Les mesures le donnent sensiblement égal à 110 ns. A titre indicatif, on donne le détail de 3 et 4 :

	3	4
4300 10 bits	50 ns	60 ns
4300 11 bits	45 ns	60 ns

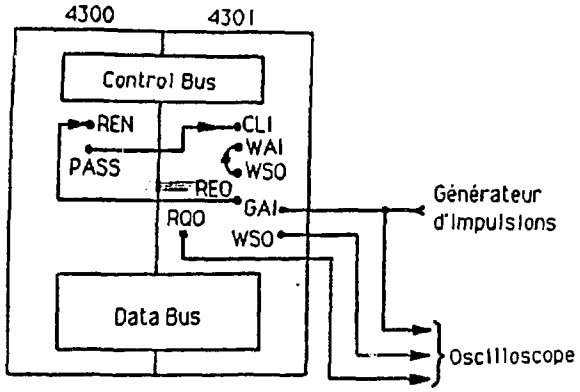


Schéma 2/1 : Vitesse de transfert sur le bus FERA.

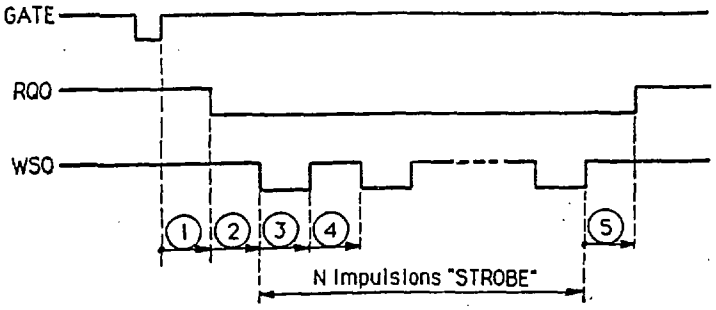


Schéma 2/2 : chronogramme obtenu sur le bus FERA.

Note : même si on lit deux voies dont les addresses ne sont pas consécutives, le temps écoulé entre les deux lectures est 110 ns.

->5 = Donné à titre indicatif : 20 ns.

Le nombre N de WSO observés dépend du type de lecture choisi : En mode sans suppression de 0, 16 WSO correspondant chacun à une donnée. En mode avec suppression de 0, N= nombre de voies >0 + 1 dû au mot de tête contenant le VSN et le "Word Count" du codeur.

* Essai des différentes configurations de STATUS.

On utilise ici la mémoire 4302 LECROY (schéma 2/3). On envoie des données en entrée du 4300 qui, après codage, charge la mémoire via le bus ECL FERA. En relisant la mémoire par CAMAC, on peut vérifier le format des données. Dans les deux modes de lecture (suppression ou pas de 0), on n'observe aucun problème que ce soit en accès CAMAC ou en lecture ECL.

Note : dans le mode suppression de 0, un codeur dont toutes les voies sont nulles, n'émet aucun signal sur le bus FERA. Il est alors transparent sur ce bus. Si la même situation se présente en lecture CAMAC, le codeur n'émet pas de LAM et sa réponse Q est inhibée.

Dans l'acquisition prévue pour les multidétecteurs au GANIL, la baie contenant les voies additionnelles, sera éloignée de quelques mètres des baies contenant les CPU d'acquisition. Si ces voies comprennent des modules LECROY 4300, le bus FERA de liaison avec les CPU sera long d'environ 15 mètres. Afin de s'assurer de la viabilité d'un tel dispositif, on modifie le montage 2/3 en portant la longueur du bus de données et celle de la ligne WSO (4301)->WSI (4302)à une vingtaine de mètres. Pour les différents modes de lecture possibles, on n'observe pas de "distorsion" des données.

* Test de la remise à zéro.

On veut vérifier l'influence de la largeur du signal CLR et de sa position par rapport à la GATE sur l'acquisition.

Le schéma du montage utilisé est donné en 2/4.

La synchronisation du générateur B sur le générateur A permet de régler la position du CLR par rapport à celle de la GATE. D'autre part, un réglage de la largeur du signal CLR est possible.

La procédure est la suivante : On laisse les entrées analogiques du 4300 "en l'air" et on déclenche le codage avec une GATE de largeur indifférente. Un programme d'acquisition vient, entre la GATE et le CLR tester le LAM émis par le codeur en fin de codage, lire une des voies du codeur pour l'afficher sur un écran puis retourne en attente de LAM.

Dans un premier temps, on fixe le retard du CLR sur la GATE pour boucler sur le diagramme suivant : GATE ->ACQUISITION ->CLR.Sachant qu'on lit toujours la même voie avec les même conditions d'entrée, on doit toujours lire la même valeur.On fait varier la largeur du signal CLR et on regarde si l'affichage se modifie. Pour des largeur de CLR variant de 1 microseconde à 20 nanosecondes, on n'observe aucune altération de la valeur lûe.

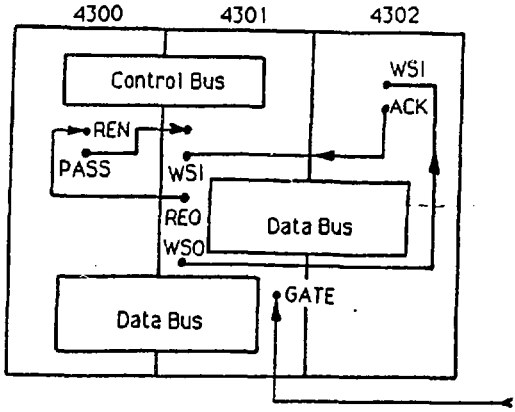


Schéma 2/3 : Test des différents modes de lecture.

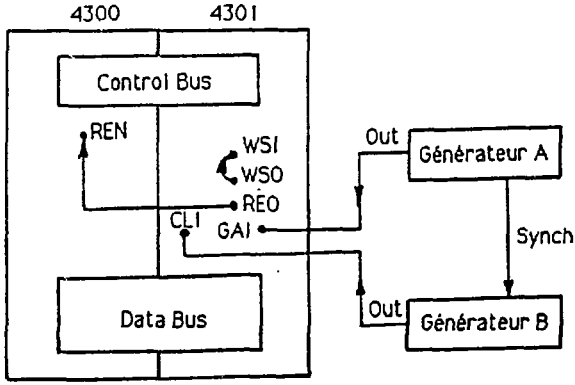


Schéma 2/4 : Test de la RAZ.

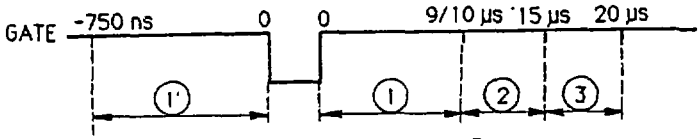


Schéma 2/5 : Test de la RAZ.

Dans un deuxième temps, on fixe la largeur du CLR sur la GATE à une valeur moyenne (100 ns par ex), et on "déplace" l'impulsion CLR autour de l'impulsion GATE Avec le codeur 4300 11 bits on obtient le diagramme de temps 2/5.

-> Si le CLR arrive pendant 1, il intervient pendant le codage, donc on n'a jamais de données converties; l'acquisition est bloquée.

-> En 2 le CLR intervient entre la fin du codage et la lecture du codeur; on ne lit donc que des 0.

-> A partir de 3, on lit un mélange de 0 et de valeurs codées.

-> Au delà, les valeurs lûes sont correctes.

-> Dans 1', on trouve des résultats abhérants.

Note : si on déconnecte la ligne CLR de l'acquisition, celle-ci s'arrête et reprend lorsqu'on reconnecte le CLR.

22/ Test du bus avec deux codeurs.

* Le montage est donné schéma 2/6.

* Les problèmes rencontrés.

Dans un premier temps, le CLR des codeur est assuré en reliant la sortie PASS du dernier codeur sur le bus (le plus proche du 4301) à l'entrée CLI du 4301 mais cette méthode pose des problèmes.

Prenons le cas où les deux codeurs fonctionnent sans suppression de 0. Le codeur 10 bits demande le bus après 5,3 microsecondes et transmet ses 16 données en environ 1,8 microsecondes,soit au total : 7,1 microsecondes. Alors, il émet le signal PASS vers le codeur 11 bits qui lui, n'a pas encore terminé de coder et de ce fait transmet le signal PASS sur le CLI du 4301. Ainsi les deux codeurs sont remis à zéro alors que le 4300 11 bits n'a pu être lû. On constate que : quelle que soit la configuration adoptée, si la lecture du codeur 10 bits se termine avant l'arrivée du signal REQ du 11 bits, ce dernier est remis à zéro avant de pouvoir être lû.

Dans un deuxième temps, le CLR vient de l'extérieur. Il est obtenu par un générateur d'impulsions synchronisé sur le générateur de GATE. En retardant suffisament (20 microsecondes par ex) le signal CLR par rapport au signal GATE, on observe alors effectivement la lecture des deux codeurs (cf chronogramme 2/7).

->1 = 5,3 ou 7,6 microsecondes suivant que l'on a ou pas suppression de 0.

->2 = 9,3 ou 11,5 microsecondes.

Si la lecture du 4300 10 bits n'est pas terminée quand le 4300 11 bits est prêt à envoyer ses données sur le bus, les deux signaux de requête de bus se chevauchent et on a alors le diagramme de temps 2/8. Le codeur 10 bits garde le bus tant qu'il a des données à émettre. Lorsqu'il a terminé, le signal PASS est transmis sur l'entrée REN du codeur 11 bits qui peut alors envoyer ses données vers le 4301. Il faut noter que le temps de passage de la lecture de la dernière donnée du 10 bits, à la lecture de la première donnée du 11 bits est encore de 110 ns.

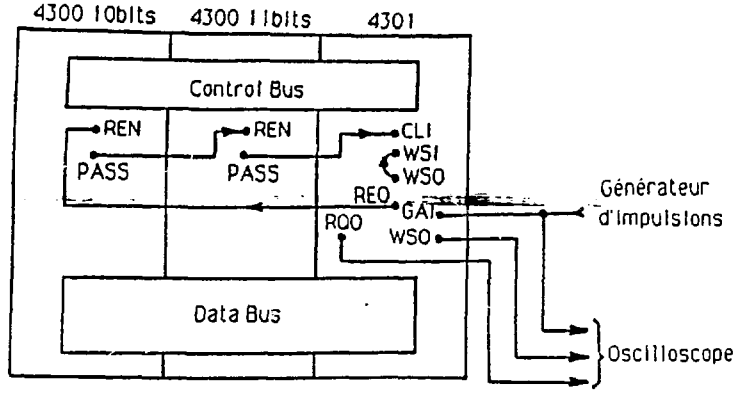


Schéma 2/6 : Test du bus FERA avec deux codeurs.

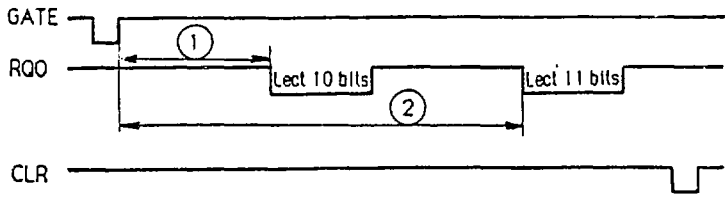


Schéma 2/7 : Lecture de deux codeurs avec un CLR extérieur.

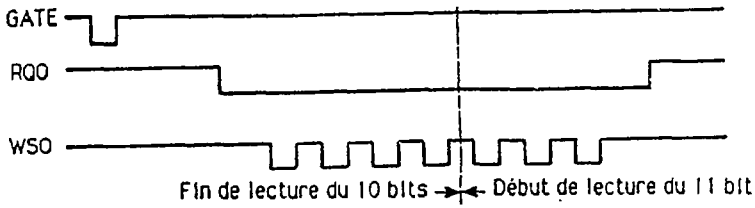


Schéma 2/8 : Lecture de deux codeurs prêts dans des temps identiques.

* La configuration prévue au GANIL.

Dans l'acquisition des multidétecteurs XYZT et DELF, on aura des QDC et des ADC sur le même bus FERA. Les temps de conversion de ces codeurs sont respectivement : 11,5 microsecondes pour les QDC (quel que soit le nombre de voies codées) et 2 à 3 microsecondes par voies pour les ADC. Pour limiter les temps morts, on prévoit un découpage de la DAISY-CHAIN sur le bus FERA, en deux secteurs, ce qui permettra de lire les QDC avant la fin de codage des ADC.

2.3/ Test de la ligne INHIBIT READOUT.

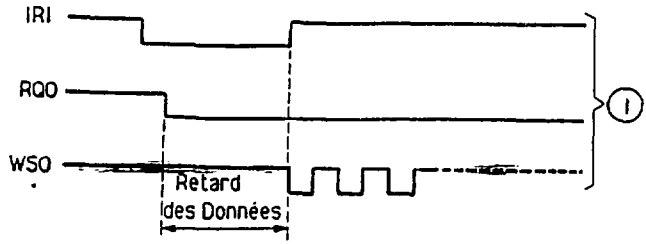
Une solution possible aux problèmes ci-dessus est d'utiliser la ligne IRI permettant d'inhiber la lecture des données tout le temps qu'elle est activée. Ainsi, on pourrait retarder la lecture des ADC jusqu'à ce que ceux-ci soient prêts par exemple. Les deux utilisation possibles de cette ligne sont données en 2/9.

conclusion.

Informés des performances en linéarité différentielle du codeur 4300, les futurs utilisateurs les ont considérées comme acceptables, à condition que des garanties soient prise dès la commande, auprès de LECROY pour que la linéarité différentielle n'excède pas +/- 1%.

L'étude du bus FERA montre que celui-ci est mal adapté à l'utilisation de codeurs de type différent. Il faudra donc ajouter une logique extérieure (peut-être réduite à de simples retards) pour séquencer correctement les lectures. D'autre part, il apparait qu'en mode "sans suppression de 0", on ne dispose d'aucune information sur la provenance des données. Il est donc recommandé, même si cela "coûte" 2,3 microsecondes de plus, de fonctionner en mode "suppression de 0".

L'avantage majeur de ces codeurs est que la lecture est séquencée par le "matériel", donc elle est rapide et les CPU d'acquisition peuvent se consacrer uniquement au traitement des données. De plus, on dispose encore de la possibilité de contrôler le fonctionnement de l'ensemble par lecture CAMAC.



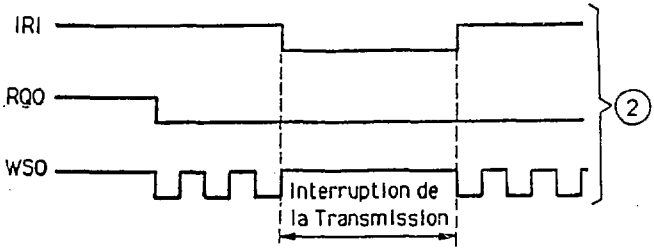


Schéma 2/9 : Utilisation de la ligne IRI.

ANNEXE 2

Le bus VME : description de la partie matérielle et principales caractéristiques

Annoncé à Munich par Motorola, Mostek, Philips/Signetics et Thomson-Efcis, à l'occasion de Systems 81 (voir « minis et micros », n°151), le bus VME, extensible à 32 bits pour les adresses et les données, peut représenter face à d'autres références comme le Multibus, un pas important vers la standardisation des cartes double Europe. Dans ce premier article, nous abordons la description du bus VME et dans un second, nous en décrirons le fonctionnement.

Le VME (Versa Module Eurocard) est une nouvelle structure de bus pour les systèmes 8, 16 et 32 bits. C'est un bus multi-microprocesseur, permettant à plusieurs cartes-maîtres, de partager des ressources communes (mémoire, entrées/sorties) suivant une priorité d'accès parallèle ou tournante.

La structure du bus VME; qui dispose de quatre niveaux d'accès, est construite autour du concept « maître-esclave » : le maître a le contrôle du bus, tandis que l'esclave, après décodage de l'adresse le concernant, répond à la commande envoyée par le maître. Un cycle indivisible de lecture/modification/écriture permet l'utilisation de sémaphores pour se réserver une ressource commune dans un environnement multi-microprocesseur. De plus, sept niveaux d'interruption peuvent être centralisés ou répartis parmi les différents processeurs. Enfin, la nature asynchrone du bus permet de faire cohabiter des processeurs, des mémoires et des périphériques présentant des caractéristiques de vitesses différentes.

Le bus VME est implanté sur des cartes au format double Europe, satisfaisant aux normes mécaniques DIN 41612 et DIN 41494.

Une structure similaire au Versabus

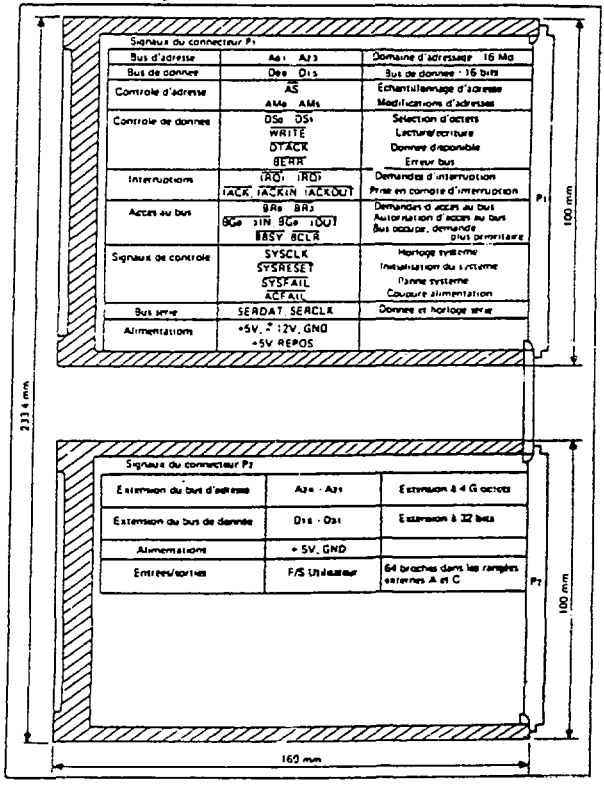
Electriquement, le bus se répartit sur deux connecteurs P1 et P2 de 96 broches (fig. 1).

Le connecteur P1 dispose d'un bus de données de 16 bits, d'un bus d'adresses de 24 bits, de six lignes de modification d'adresses et de diverses lignes d'alimentation et de contrôle : demandes d'accès au bus, prises en compte de la demande d'accès au bus, demandes d'interruption, prises en compte de la demande d'interruption. En plus du bus parallèle, le VME possède un canal de communication série qui assure, en fonctionnement multi-microprocesseur, des échanges de signaux de service ou d'alarme. Il est ainsi possible d'optimiser les transferts sur le bus parallèle.

Le connecteur P2 contient les lignes d'extension à 32 bits des bus de données et d'adresses. Ces extensions, destinées aux futurs microprocesseurs 32 bits, permettent des transferts 32 bits entre la mémoire et des coupleurs performants, tels que les coupleurs de disques.

Le bus VME reprend beaucoup de concepts de Versabus, développé par Motorola pour l'Exormacs. Pres-▸

Fig. 1 - Les connecteurs à 96 broches du bus VME.



▸que tous les signaux sont identiques et présentent le même chronogramme. Le bus peut donc être considéré comme un sous-ensemble du Versabus. Néanmoins, il existe de nombreuses différences, essentiellement parce qu'il a fallu supprimer quelques lignes du Versabus pour faire tenir, sur deux connecteurs de 96 broches, ce qui existe sur deux connecteurs de 120 broches.

Les lignes d'alimentation ± 15 V d'horloge AC, et de parité de bus d'adresses et de bus de données ont été supprimées. Deux lignes de contrôle de test, un niveau de priorité d'accès au bus, ainsi que plusieurs lignes de modification d'adresses ont de plus été éliminées.

Le bus VME a été conçu initialement avec le souci de faire coexister des cartes au format simple et double Europe. C'est pourquoi tous les signaux nécessaires à la conception d'un système 16 bits sont implantés sur le connecteur P1, permettant, en particulier, la réalisation d'un système compact. Le bus VME offre de nombreuses possibilités de réalisation, aussi bien de systèmes monoprocesseurs compacts que de systèmes multiprocesseurs très élaborés.

ORGANISATION DU BUS

Les fonctions d'interface du bus sont au nombre de quatre : transfert de données, demandes d'accès au bus, gestion des interruptions et signaux de contrôle. Nous allons les étudier successivement.

Transfert de données

Le transfert des données s'effectue par l'intermédiaire d'un bus spécialisé (DTB), comprenant le bus d'adresses, le bus de données et les signaux de contrôle, associés à ce transfert. Des coupleurs specialisés, tant sur les cartes-maîtres qu'esclaves assurent l'interface avec le bus DTB qui se compose des lignes suivantes :

— le bus d'adresses A01 à A31 ;
— les lignes de modification d'adresses AM0 à AM5 ;
— le bus de données D00 à D31 ;
— les lignes de contrôle de transfert par le maître : AS (échantillonnage d'adresse) ; DS0 (sélection de l'octet impair — octet de poids faible — à l'intérieur d'un mot de 16 bits) ; DS1 (sélection de l'octet pair — octet de poids fort — à l'intérieur d'un mot de 16 bits) ; LWORD (accès à un double mot) ; WRITE (sélection lecture/écriture) ;
— les lignes d'état (contrôlées par l'esclave) avec DTACK (donnée disponible c'est-à-dire transfert réussi) ; BERR (erreur bus révélant un problème avec le cycle en cours).

Lignes de modification d'adresse

Avec ces lignes, le maître peut envoyer des informations supplémentaires lors d'un transfert de données, informations qui spécifient diverses fonctions :

— **Configuration dynamique du système** : chaque carte esclave dans le système est configurable de manière à ne répondre qu'à un seul code, généré sur les lignes de modification d'adresse. S'il y a plusieurs

Tableau I
Codes de modification d'adresse

Lignes de modification d'adresse						Fonctions	Défini par
AM 5	AM 4	AM 3	AM 2	AM 1	AM 0		
1	1	1	1	1	1	Accès ascendant en mode superviseur standard	Spécifié Bus VME
1	1	1	1	1	0	Accès au programme en mode superviseur standard	Spécifié Bus VME
1	1	1	1	0	1	Accès aux données en mode superviseur standard	Spécifié Bus VME
1	1	1	1	0	0	Non défini	Réservé
1	1	1	0	1	1	Accès ascendant en mode utilisateur standard	Spécifié Bus VME
1	1	1	0	1	0	Accès au programme en mode utilisateur standard	Spécifié Bus VME
1	1	1	0	0	1	Accès aux données en mode utilisateur standard	Spécifié Bus VME
1	1	1	0	0	0	Non défini	Réservé
1	1	0	X	X	X	Non défini	Réservé
1	0	1	1	1	1	Non défini	Réservé
1	0	1	1	1	0	Non défini	Réservé
1	0	1	1	0	1	Accès entrée/sortie en mode superviseur court	Spécifié Bus VME
1	0	1	1	0	0	Non défini	Réservé
1	0	1	0	1	1	Non défini	Réservé
1	0	1	0	1	0	Non défini	Réservé
1	0	1	0	0	1	Accès entrée/sortie en mode utilisateur court	Spécifié Bus VME
1	0	1	0	0	0	Non défini	Réservé
1	0	0	X	X	X	Non défini	Réservé
0	1	X	X	X	X	Non défini	Utilisateur
0	0	1	1	1	1	Accès ascendant en mode superviseur étendu	Spécifié Bus VME
0	0	1	1	1	0	Accès au programme en mode superviseur étendu	Spécifié Bus VME
0	0	1	1	0	1	Accès aux données en mode superviseur étendu	Spécifié Bus VME
0	0	1	1	0	0	Non défini	Réservé
0	0	1	0	1	1	Accès ascendant en mode utilisateur étendu	Spécifié Bus VME
0	0	1	0	1	0	Accès au programme en mode utilisateur étendu	Spécifié Bus VME
0	0	1	0	0	1	Accès aux données en mode utilisateur étendu	Spécifié Bus VME
0	0	1	0	0	0	Non défini	Réservé
0	0	0	X	X	X	Non défini	Réservé

Notes :
— les codes spécifiés bus VME ne peuvent être utilisés pour des fonctions autres que celles spécifiées ;
— les codes destinés à l'utilisateur sont à l'entière disposition de celui-ci ;
— les codes « réservés » ne peuvent être employés par l'utilisateur ; ils sont réservés pour l'utilisation système et pour de futures extensions ;
— le mode court utilise les lignes d'adresse A01-A15 ;
— le mode standard utilise les lignes d'adresse A01-A28 ;
— le mode long utilise les lignes d'adresse A01-031 ;
— « Accès Ascendant » signifie accès séquentiel de la mémoire.

maîtres sur le bus, il peut être intéressant d'allouer à chaque maître un code d'accès particulier à cet esclave (chaque maître a un code différent pour accéder au même esclave). Une telle précaution évite une panne générale en cas de mauvais fonctionnement d'une carte maître.

— **Choix de l'emplacement d'un esclave à l'intérieur de l'espace d'adressage d'un maître :** n'importe quelle carte esclave peut répondre à des adresses différentes, suivant le code généré sur les lignes de modification d'adresse. Une telle caractéristique permet à chaque maître de sélectionner dynamiquement, à l'intérieur de son espace d'adressage, l'emplacement d'un esclave.

— Accord de privilèges d'adressage à un esclave : par l'intermédiaire du code genere sur les lignes de modification d'adresses, il est possible d'établir des niveaux privilégiés d'accès à un esclave ; le code indique dans ce cas le niveau de privilège.

— Envoi d'un code de fonction : en plus des codes générés par l'utilisateur, les lignes de modification d'adresses servent à indiquer le type de cycle de bus (tableau I). Les codes de fonction, réservés à la définition du cycle de bus, ne peuvent être utilisés par le programmeur.

— Sélection d'un système de gestion de mémoire parmi plusieurs : chaque système de gestion de mémoire transforme des segments logiques en segments physiques (traduction d'adresses logiques en adresses physiques). A chaque tâche *est alloué un certain nombre* de segments. Lorsque le superviseur passe d'une tâche à une autre, il doit modifier les contenus des registres de segments du système de gestion de mémoire ou commuter un système de gestion à un autre. Dans ce dernier cas, qui est plus performant, il est alloué, à chaque tâche, un système de gestion de mémoire. La sélection d'un système de gestion de mémoire parmi plusieurs se fait par les lignes de modification d'adresses.

Bus de données

Un transfert de données s'effectue au niveau de l'octet, du mot de 16 bits ou du mot de 32 bits.

Le transfert d'un mot de 16 bits s'effectue sur les lignes D00 à D15. L'adresse du mot est alignée sur l'adresse de l'octet pair (octet de poids fort véhiculé sur D15-D08 alors que l'impair, octet de poids faible, est transféré sur D07-D00). La sélection d'un octet à l'intérieur d'un mot s'effectue par les lignes de sélection DS0 et DS1.

LWORD	A01	DS0	DS1	Type de transfert
0	1	X	X	Illégal
0	0	0	0	Transfert d'un double mot (32 bits)
0	0	0	1	Illégal
0	0	1	0	Illégal
0	0	1	1	Illégal
1	1	0	0	Transfert d'un mot impair
1	1	0	1	Transfert de l'octet impair d'un mot impair
1	1	1	0	Transfert de l'octet pair d'un mot impair
1	1	1	1	Illégal
1	0	0	0	Transfert d'un mot pair
1	0	0	1	Transfert de l'octet impair d'un mot pair
1	0	1	0	Transfert de l'octet pair d'un mot pair
1	0	1	1	Illégal

Tableau II
Types de transfert sur le bus de données

Le transfert d'un double mot (32 bits) se fait sur les lignes D00 à D31. L'adresse du double mot est alignée sur l'adresse du mot pair (A01 = 0) qui est le plus significatif et véhiculé sur D16-D31, le mot impair étant le moins significatif et étant transféré sur D00-D15.

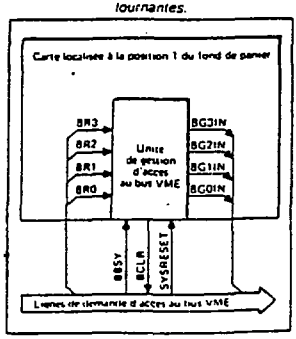
Le tableau II résume les différentes possibilités de transfert sur le bus de donnée.

Demandes d'accès au bus

Le bus VME dispose de quatre niveaux de priorité d'accès au bus, *assignés statiquement (hiérarchie de* priorité fixe) ou dynamiquement (hiérarchie de priorité tournante) à des lignes spécialisées BR3 - BR0 (Bus Request). Le niveau de priorité d'une carte-maître dépend du niveau de priorité $\overline{BR}$ à laquelle cette carte est connectée et de sa position à l'intérieur du châssis.

Explicitons ce mode de fonctionnement : chaque ligne $\overline{BR}$ est connectée à des maîtres par des « OU câblés ». De cette manière, plusieurs maîtres peuvent partager la même ligne $\overline{BR}$. A chaque ligne $\overline{BR}$ correspond une ligne $\overline{BG}$ d'autorisation d'accès au bus qui relie en « daisy chain » tous les maîtres associés à la ligne $\overline{BR}$ et alloue le bus par simple priorité sérielle, établissant ainsi un deuxième niveau de priorité suivant la position de la carte dans le châssis. En option, le bus VME peut fonctionner sur un seul niveau de priorité $\overline{BR3}$.

Fig. 2 - Unité de gestion d'accès au bus DTB avec priorité d'accès fixes. La ligne $\overline{BCLR}$ est supprimée dans le cas de priorités tournantes.



Unité de gestion d'accès au bus de transfert

Cette unité de gestion est localisée dans la position 1 du châssis. Elle établit une hiérarchie des demandes d'accès au bus et accorde le bus à la demande la plus prioritaire. Deux cas sont à considérer suivant que la hiérarchie des niveaux d'accès au bus est fixe (fig. 2) ou tournante.

1. Hiérarchie fixe : l'unité de gestion reçoit les quatre lignes de demande d'accès au bus ($\overline{BR0}$-$\overline{BR3}$) auxquelles sont alloués des niveaux de priorité fixe : la hiérarchie des priorités va, dans l'ordre décroissant, de $\overline{BR3}$ à $\overline{BR0}$. Elle alloue le bus à la demande la plus prioritaire et génère un niveau bas sur la broche d'autorisation d'accès au bus $\overline{BGIN}$ correspondante. Notons que l'autorisation d'accès au bus de transfert est conditionnée par la disponibilité de ce bus, signalée par un niveau haut sur la broche $\overline{BBSY}$. La broche $\overline{BCLR}$ de l'unité de gestion signale au maître, en possession du bus, qu'une demande d'accès plus prioritaire est en attente.

2. Hiérarchie tournante : l'unité de gestion reçoit les quatre lignes de demande d'accès au bus VME ($\overline{BR0}$-$\overline{BR3}$). Le niveau alloué à chaque ligne n'est plus fixe, mais tournant ; le maître, qui a le niveau de priorité le plus élevé, a, après accès au bus, le niveau de priorité le plus bas. Mis à part la différence de fonctionnement au niveau des priorités, les unités de gestion d'accès au bus sont semblables à part le fait que la ligne $\overline{BCLR}$ n'

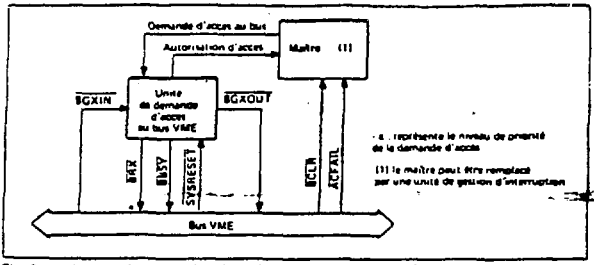


Fig. 3 - Logique de demande d'accès au bus dans le cas où le maître restitue le bus immédiatement après usage.

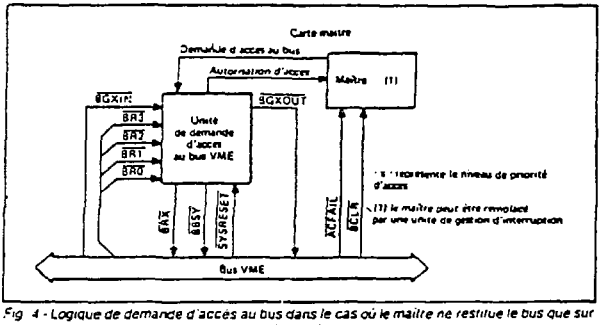
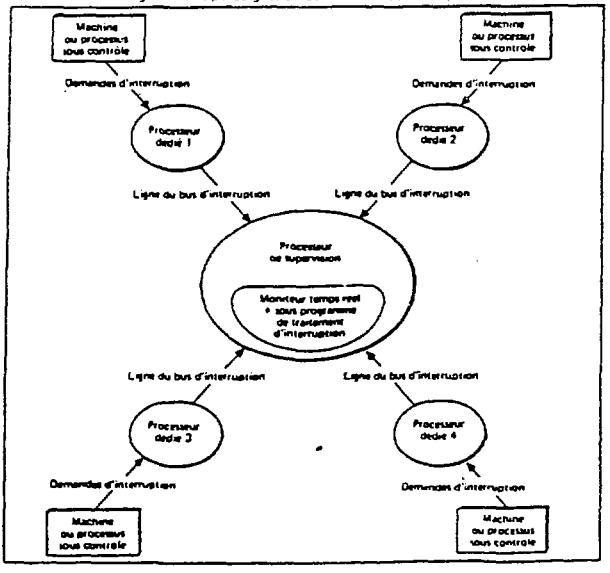


Fig. 4 - Logique de demande d'accès au bus dans le cas où le maître ne restitue le bus que sur demande.

Fig. 5 - Principe de gestion centralisée des interruptions.



est supprimée dans le mode avec priorité tournante.

Unité de demande d'accès au bus VME

Chaque maître possède une unité de demande d'accès au bus VME qui peut fonctionner suivant deux modes (fig. 3) et fig. 4) : l'un dit RWD (Release when done) dans lequel le maître libère le bus dès qu'il n'en a plus besoin et l'autre dit ROR (Release on request) dans lequel le maître ne restitue le bus, après l'avoir utilisé, que si une demande en est faite, sur une des broches BR0-BR3.

Cette dernière approche permet de diminuer le nombre de demandes générées par un maître, qui constituent un pourcentage non négligeable du trafic sur le bus.

La broche BBSY est portée à l'état bas dès que le maître prend en charge le bus VME. La libération du bus, signalée par un état haut sur la broche BBSY, a lieu à des moments différents suivant le mode de fonctionnement adopté.

Le maître doit gérer deux lignes supplémentaires dans le cadre d'une demande d'accès au bus VME. Ces lignes, ACFAIL et BCLR, indiquent au maître, en possession du bus, que des demandes d'accès au bus plus prioritaires sont en attente. ACFAIL signale qu'une chute d'alimentation du secteur a été détectée ; dans ce cas, la maître doit rendre le bus quel que soit le transfert qu'il est en train d'exécuter. BCLR indique une demande plus prioritaire. Dans ce cas, le maître détermine, suivant le type de transfert qu'il est en train d'exécuter, l'instant où il libérera le bus.

Gestion des interruptions

Deux types de système de gestion des interruptions peuvent être envisagés :

— un système de gestion centralisée qui confie à un seul processeur le traitement des demandes d'interruption ;

— un système de gestion distribuée qui répartit sur plusieurs processeurs les demandes d'interruption et confie à chacun d'eux le traitement de celles-ci.

Système de gestion centralisée des interuptions

La figure 5 donne le principe d'un système de gestion centralisée des interruptions. Une telle structure est bien adaptée aux applications de contrôle de processus industriel : chaque processeur dédié assure le

contrôle d'une machine ou d'un processus et ne peut tolérer qu'un nombre minimal d'interruptions. Le traitement de celles-ci est regroupé sur un processeur de supervision, qui prend en compte chaque demande d'interruption suivant une hiérarchie de priorités. Les processeurs dédiés reçoivent les demandes d'interruptions, générées par les machines ou par les processus, mais ne les traitent pas.

Systèmes de gestion décentralisée des interruptions

Les principes en sont illustrés par la figure 6. Une telle structure est bien adaptée aux applications de traitement par lots (« batch »). Avec ce type d'organisation, le traitement de la tâche prochaine est confié au prochain processeur disponible. Chaque processeur du système exécute une partie du système d'exploitation. Il ne prend en compte que les demandes d'interruptions qui lui sont adressées par les autres processeurs du système. Comme le traitement de certaines interruptions exigent l'accès à des ressources globales, il est nécessaire que la partie du système d'exploitation, incluse dans chaque processeur, puisse dialoguer avec le système de gestion des ressources communes.

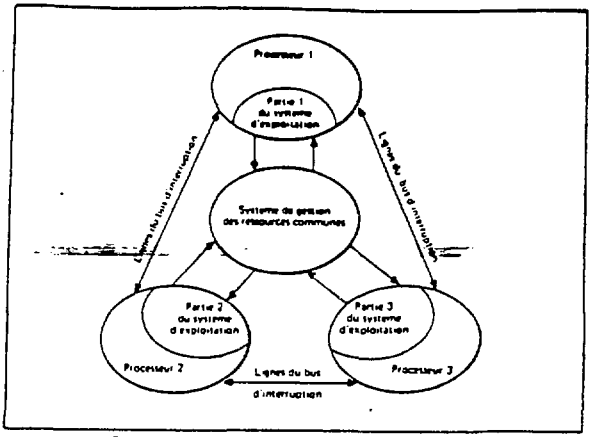


Fig. 6 - Principe de gestion décentralisée des interruptions

Signaux utilisés dans les demandes d'interruption

Les lignes du système de gestion des demandes d'interruption comprennent les sept lignes de demande d'interruption IRQ7 à IRQ1, une ligne de prise en compte d'une demande d'interruption IACK, deux lignes IACKIN et IACKOUT, établissant une priorité sérielle (daisy chain) de la prise en compte des demandes d'interruption.

Chaque ligne de demande d'interruption est rendue active par un niveau bas généré par l'unité de demande d'interruption (Interrupter). Toutes les cartes, susceptibles de générer des demandes d'interruption disposent d'une telle unité. Dans le cas d'un système de gestion centralisée d'interruptions, chaque ligne d'interruption a un niveau de priorité (IRQ7 a le niveau de priorité le plus élevé).

Chaque ligne d'interruption peut recevoir des demandes d'interruption provenant de plusieurs sources. La détermination de la source la plus prioritaire se fait par simple priorité sérielle (daizy chain) à partir de la ligne de prise en compte IACK. Cette ligne, connectée à la broche IACKIN du connecteur A1 du fond du panier,

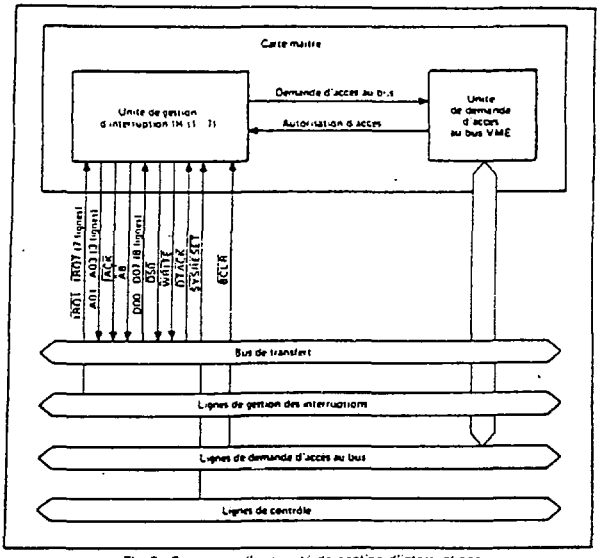


Fig. 7 - Connexion d'une unité de gestion d'interruptions

relie en « daisy chain », les différentes sources d'interruptions (unités de demande d'interruption), par l'intermédiaire des broches IACKIN/IACKOUT.

La broche IACKOUT de la source la plus prioritaire est reliée à la broche IACKIN de la source dont le niveau de priorité est juste en dessous, et ainsi de suite, jusqu'à la source dont le niveau de priorité est le plus bas. La broche IACKIN de la source la plus prioritaire est reliée à la broche IACKIN du connecteur A1 du fond du panier. Lorsque la source la plus prioritaire n'est pas à l'origine de l'interruption, elle envoie à la prochaine source, à travers IACKOUT, un niveau bas qui est reçu sur la bro-

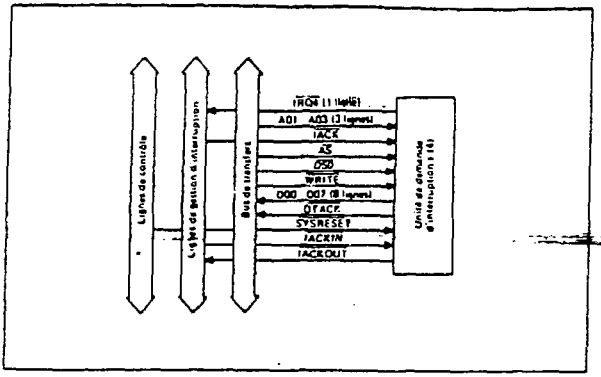


Fig. 8 - Connexion d'une unité de demande d'interruptions

▸ che IACKIN et ainsi de suite. Parmi les sources qui ont généré une demande d'interruption, celle qui a le niveau le plus élevé génère sur sa broche IACKOUT un état bas, inhibant ainsi l'accès au bus par les circuits suivants dans la chaîne.

Unité de gestion *des interruptions* (Interrupt Handler)

La gestion des interruptions peut être centralisée sur un seul processeur spécialisé ou répartie sur plusieurs processeurs. Quel que soit le type de gestion adopté, les rôles d'une unité de gestion sont les suivants :

— elle détermine, parmi les demandes d'interruption qu'elle a reçues, celle à qui le niveau de priorité est le plus élevé ;

— elle utilise l'unité de demande d'accès au bus, qui lui est associée, pour faire une demande d'accès et, si cette dernière est accordée, elle génère un signal de prise en compte IACK ;

— elle lit le vecteur d'interruption et initialise la séquence de traitement de l'interruption.

Une unité de gestion dans un système est caractérisé par le groupe d'interruptions qu'elle gère. L'expression IH (*a-b*) indique la ligne « a » dont le niveau d'interruption est le plus élevé. En plus des lignes d'interruption « *a* » et «*b* », l'unité de gestion a en charge toutes les lignes d'interruption de niveau intermédiaire. La figure 8 représente une unité de gestion IH (1-7), ainsi que les broches associées à cette unité de gestion.

Unité de demande d'interruption

L'unité de demande d'interruption, associée à chaque carte génératrice d'interruptions, assure les fonctions suivantes :

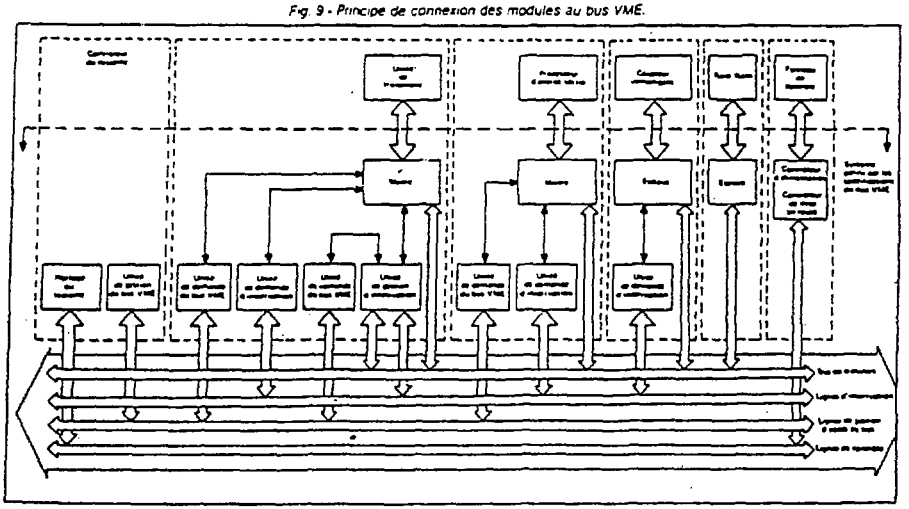
— elle adresse une demande d'interruption à l'unité de gestion d'interruption, associée à la ligne de demande d'interruption utilisée ;

— elle fournit un vecteur d'interruption, lorsqu'elle reçoit un signal de prise en compte de sa demande d'interruption ;

— elle transmet le signal de prise en compte, si elle n'est pas la source de la demande d'interruption.

Une unité de demande d'interruption est spécifiée par la ligne de *demande d'interruption* qui lui est associée. La caractéristique I (*n*) spécifie l'unité de demande d'interruption dont la ligne d'interruption associée est IRQ *n*. La figure 8 représente une unité d'interruption I (4) dont la ligne de demande d'interruption est IRQ4.

Fig. 9 - Principe de connexion des modules au bus VME.



Noms des broches	Rangée A	Rangée B	Rangée C
1	D00	BBSY	D08
2	D01	BCLR	D09
3	D02	ACFAIL	D10
4	D03	BG0IN	D11
5	D04	BG0OUT	D12
6	D05	BG1IN	D13
7	D06	BG1OUT	D14
8	D07	BG2IN	D15
9	GND	BG2OUT	GND
10	SYSCLK	BG3IN	SYSFAIL
11	GND	BG3OUT	BERR
12	DS1	BR0	SYSRESET
13	DS0	BR1	LWORD
14	WRITE	BR2	AM5
15	GND	BR3	A23
16	DTACK	AM0	A22
17	GND	AM1	A21
18	AS	AM2	A20
19	GND	AM3	A19
20	IACK	GND	A18
21	IACKIN	SERCLK (1)	A17
22	IACKOUT	SERDAT (1)	A16
23	AM4	GND	A15
24	A07	IRQ7	A14
25	A06	IRQ6	A13
26	A05	IRQ5	A12
27	A04	IRQ4	A11
28	A03	IRQ3	A10
29	A02	IRQ2	A09
30	A01	IRQ1	A08
31	— 12 V	+ 5 V STDBY	+ 12 V
32	+ 5 V	+ 5 V	+ 5 V

Note : SERCLK et SERDAT sont prévues pour l'implantation d'un bus de communication série.

Tableau III
Implantation des signaux sur le connecteur P1

Tableau IV
Implantation des signaux sur le connecteur P2

Numéro des broches	Rangée A	Rangée B	Rangée C
1	E/S utilisateur	+ 5 V	E/S utilisateur
2	"	GND	"
3	"	Réservé	"
4	"	A24	"
5	"	A25	"
6	"	A26	"
7	"	A27	"
8	"	A28	"
9	"	A29	"
10	"	A30	"
11	"	A31	"
12	"	GND	"
13	"	+ 5 V	"
14	"	D16	"
15	"	D17	"
16	"	D18	"
17	"	D19	"
18	"	D20	"
19	"	D21	"
20	"	D22	"
21	"	D23	"
22	"	GND	"
23	"	D24	"
24	"	D25	"
25	"	D26	"
26	"	D27	"
27	"	D28	"
28	"	D29	"
29	"	D30	"
30	"	D31	"
31	"	GND	"
32	"	+ 5 V	"

Signaux de contrôle

Ces signaux comprennent :

— ACFAIL : coupure de l'alimentation « secteur » ;

— SYSRESET : initialisation du système suite à une mise en route ;

— SYSCLK : horloge du système ;

— SYSFAIL : panne du système.

ACFAIL et SYSRESET sont fournis par la carte de contrôle de l'alimentation. SYSCLK est une horloge 16 MHz avec un facteur de mérite (duty cycle) de 50 %. L'élaboration de SYSCL se fait normalement sur la carte de contrôle du système, localisée dans la position 1 du fond de panier. SYSFAIL donne le résultat d'une séquence de test, lancée immédiatement après la mise en route. Cette broche, à l'état bas pendant toute la séquence de test, passe à l'état haut, dès que ce test est terminé, à condition qu'aucune panne n'ait été détectée.

Les tableaux III et IV donnent les implantations des signaux sur les connecteurs P1 et P2 ; la figure 9 illustre le principe de connexion de divers modules au bus VME.

Dominique Girod