L'approche fonctionnelle dans la vérification des systèmes informatiques : proposition d'un ensemble de méthodologies

Mohamed Moalla

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
Spécialité Génie-Informatique

par

Mohamed MOALLA

«L'APPROCHE FONTIONNELLE DANS LA VERIFICATION DES SYSTEMES INFORMATIQUES.

PROPOSITION D'UN ENSEMBLE DE METHODOLOGIES»

Thèse soutenue le 22 décembre 1976 devant la Commission d'Examen

Président	:	L. BOLLIET
Examinateurs		R. BEAUFILS
		G. SAUCIER
Invités		J. LEBRUN
		D. FEUERSTEIN

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : M. Philippe TRAYNARD

Vice-Président : M. Pierre-Jean LAURENT

PROFESSEURS TITULAIRES

MM. BENOIT Jean	Radioélectricité
BESSON Jean	Electrochimie
BLOCH Daniel	Physique du solide
BONNETAIN Lucien	Chimie Minérale
BONNIER Etienne	Electrochimie et Electrométallurgie
BOUDOURIS Georges	Radioélectricité
BRISSONNEAU Pierre	Physique du solide
BUYLE-BODIN Maurice	Electronique
COUMES André	Radioélectricité
DURAND Francis	Métallurgie
FELICI Noël	Electrostatique
FOULARD Claude	Automatique
LESPINARD Georges	Mécanique
MOREAU René	Mécanique
PARIAUD Jean-Charles	Chimie-Physique
PAUTHENET René	Physique du solide
PERRET René	Servomécanismes
POLOUJADOFF Michel	Electrotechnique
SILBER Robert	Mécanique des Fluides

PROFESSEUR ASSOCIE

M. ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM. BLIMAN Samuel	Electronique
BOUVARD Maurice	Génie Mécanique
COHEN Joseph	Electrotechnique
LACOUME Jean-Louis	Géophysique
LANCIA Roland	Electronique
ROBERT François	Analyse numérique
VEILLON Gérard	Informatique Fondamentale et Appliquée
ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM. ANCEAU François	Mathématiques Appliquées
CHARTIER Germain	Electronique
GUYOT Pierre	Chimie Minérale
IVANES Marcel	Electrotechnique
JOUBERT Jean-Claude	Physique du solide
MORET Roger	Electrotechnique Nucléaire
PIERRARD Jean-Marie	Mécanique
SABONNADIERE Jean-Claude	Informatique Fondamentale et Appliquée
Mme SAUCIER Gabrièle	Informatique Fondamentale et Appliquée

MAITRE DE CONFERENCES ASSOCIE

M. LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

MM. FRUCHART Robert	Directeur de Recherche
ANSARA Ibrahim	Maître de Recherche
CARRE René	Maître de Recherche
DRIOLE Jean	Maître de Recherche
MATHIEU Jean-Claude	Maître de Recherche
MUNIER Jacques	Maître de Recherche

A mes parents.

A mes amis.

Je voudrais exprimer ma reconnaissance à Monsieur L. BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble, pour l'intérêt qu'il a porté à ce travail et la confiance qu'il m'a toujours témoignée et qui ont été pour moi les plus précieux encouragements.

Je suis également reconnaissant à Madame G. SAUCIER, Maître de Conférences à l'Institut National Polytechnique de Grenoble, pour m'avoir accueilli dans son équipe et suivi constamment l'accomplissement de ce travail.

Je suis très honoré de la présence de Monsieur R. BEAUFILS, Professeur à l'Université Paul Sabatier de Toulouse, qui a bien voulu accepter de faire partie du jury malgré ses nombreuses occupations.

Monsieur J. LEBRUN, Chef du Service Diagnostics au Département Grands et Moyens Systèmes de la CII-HB, m'a apporté une aide précieuse dans la compréhension des problèmes de test dans l'environnement industriel, et a contribué efficacement à l'amélioration de la rédaction de ce document. Qu'il veuille bien accepter mes sincères remerciements.

Je remercie Monsieur D. FEUERSTEIN, Chargé du Département Support Informatique pour les Communications au CNET, pour l'intérêt qu'il a accordé à ce travail et qu'il concrétise par sa participation à ce jury.

Je tiens aussi à remercier vivement mes amis de l'équipe "Conception et Sécurité des Systèmes Logiques" avec qui j'ai collaboré et, en particulier, J. SIFAKIS pour les nombreuses critiques et suggestions qui ont jalonné ce travail.

Je ne puis ne pas souligner la patience et les compétences de Mesdames G. DUFFOURD et M.J. DOREL qui ont produit ce texte, de Monsieur D. IGLESIAS et du service de reprographie qui ont assuré la réalisation délicate de ce document.

SOMMAIRE

CHAPITRE II - PROBLÈME DU TEST DES MODULES D'INTERFACE

Chapitre III - Test des Sous-Systèmes Périphériques

Chapitre IV - Le Programme COSU

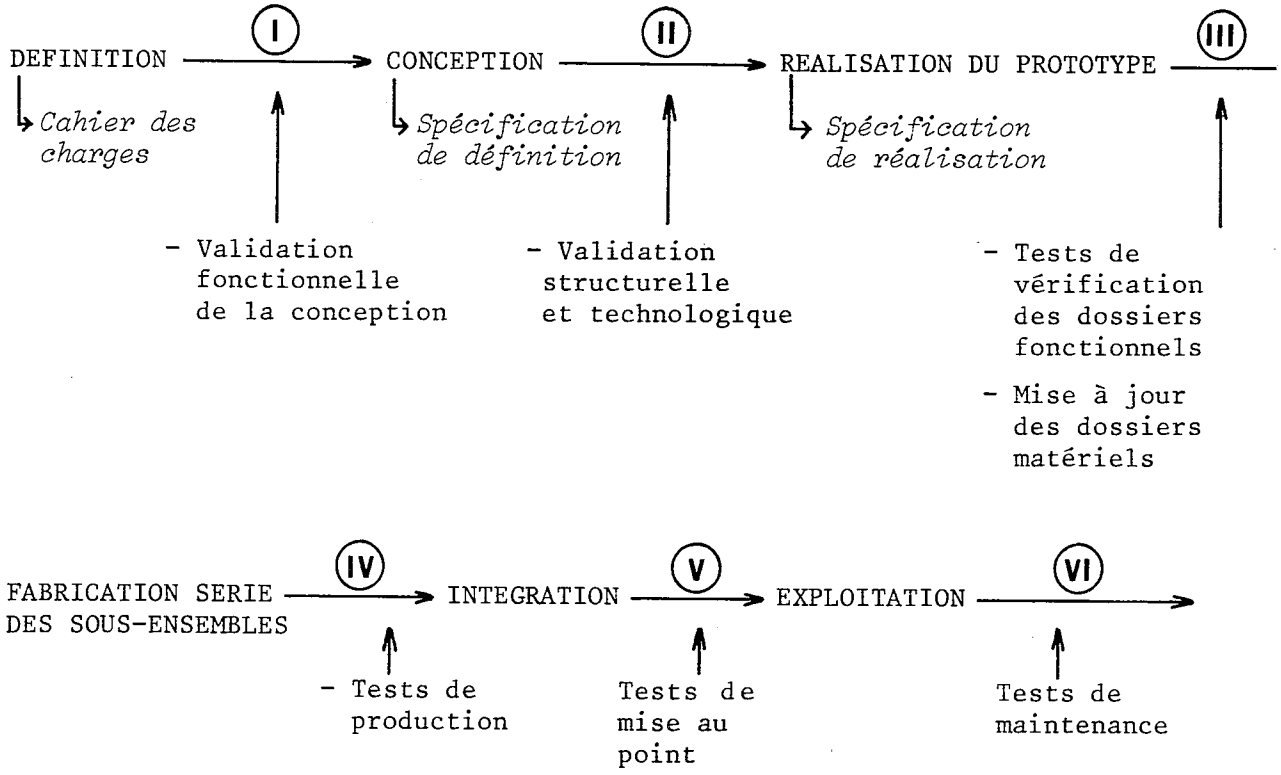
CHAPITRE V - TESTS DE VÉRIFICATION FONCTIONNELLE EN INTÉGRATION DES SYSTÈMES TESTS D'ACCEPTATION

CHAPITRE VI - OUTIL LOGICIEL SUPPORT DES TESTS D'ACCEPTATION EN INTÉGRATION DES SYSTÈMES - LE PROGRAMME SIMUX

CHAPITRE VII - VALIDATION FONCTIONNELLE EN COURS DE CONCEPTION. LE SYSTEME M.A.S.

INTRODUCTION

Le processus de passage du stade de projet initial au stade d'exploitation d'un système comporte un certain nombre d'étapes; chaque étape nécessite des moyens de vérification et de test appropriés.



La réalisation et la mise au point du prototype conduisent à l'élaboration finale de deux types de dossiers :

Les *dossiers fonctionnels* qui décrivent les possibilités fonctionnelles du système et son interface logiciel. Ils incluent :

. Les manuels d'utilisation, avec une description de l'organisation du système, des caractéristiques de fonctionnement et de performances globales.

. Les spécifications de réalisation donnant, pour chaque unité, une description plus détaillée du fonctionnement interne (structure, logigrammes fonctionnels) et de l'interface présenté, ainsi qu'une information précise des performances réalisées par l'unité.

- Les *dossiers matériels* qui décrivent les constituants du système et les interconnexions entre ces constituants. Cette description est donnée à plusieurs niveaux de détails, depuis un niveau transfert de registre assez haut jusqu'au niveau circuit. Ces dossiers comportent également une description des caractéristiques technologiques.

Des dossiers bien élaborés permettent de corréler les deux types de description à différents niveaux de détails. Cette connaissance de la correspondance fonctionnement-matériel est importante à plusieurs points de vue :

. Pour l'optimisation de l'exploitation du système en tenant compte de l'aspect matériel sous-jacent au fonctionnement des unités;

. Pour l'élaboration des tests pour la mise au point et la maintenance où, partant du matériel, on cherche les activations qui permettent de sensibiliser des chemins spécifiques;

. Pour la mise sur pied d'une politique de *suivi* (feedback) servant à améliorer la qualité des services rendus par le système.

Il va de soi que les techniques et outils utilisés pour les vérifications et tests aux diverses étapes sont différents.

En réalité, l'élaboration des dossiers fonctionnels commence dès les premières spécifications de définition du système où les objectifs de fonctionnement et de performances sont déterminés. Le but de la validation fonctionnelle dans l'étape I est de vérifier que la conception réalise ces objectifs; cette vérification est effectuée sur des *modèles*. Dans l'étape III, on effectue cette vérification sur le matériel réalisé.

Toutefois, la vérification des dossiers fonctionnels au stade du prototype ne peut être effectuée pour toutes les configurations et options possibles du système. C'est pourquoi, elle s'avère encore nécessaire en phase d'intégration (V) pour une configuration et des options particulières choisies en fonction du type d'exploitation pour laquelle le système est destiné.

Les dossiers matériels sont établis au fur et à mesure de la réalisation du prototype et ne sont définitifs qu'une fois la mise au point de ce prototype achevée. La validation des dossiers matériels en phase II a pour but de *qualifier* la conception technologique. La vérification de ces dossiers dans les phases IV, V et VI a pour but de mettre en évidence des anomalies dues respectivement à un défaut de fabrication, de réglage ou à l'usure.

Ce travail concerne dans l'ordre l'analyse et la définition des besoins de tests pour la vérification fonctionnelle durant les étapes de maintenance, mise au point en validation de prototype et en intégration série, conception du système. Pour chacun des besoins recensés, on propose une méthodologie de conception et de réalisation. L'analyse fait apparaître la nécessité d'un environnement propre à l'application de ces tests. On l'étudiera particulièrement pour le cas des tests des sous-systèmes périphériques et des tests de simultanéité au niveau système. A ce jour, l'étude et le développement de ces environnements sont restés du domaine exclusif des industriels.

Dans le premier chapitre, on situe les différents types de tests effectués le long du processus de fabrication et en maintenance, et on présente un panorama des méthodes et techniques utilisés tant pour l'élaboration de ces tests que pour leur mise en oeuvre.

L'état actuel de la recherche permet de maîtriser convenablement la résolution des problèmes de génération des séquences de test au niveau de sous-systèmes logiques synchrones et de taille réduite. Cependant, peu de travaux ont abordé de façon satisfaisante le problème de la composition de ces tests au niveau système; et en particulier celui du test des interfaces interconnectant les sous-systèmes. Ce dernier est introduit dans le chapitre II.

Les chapitres III et IV abordent le problème du test des sous-systèmes périphériques. Ces unités présentent certaines particularités :

- D'une part, par leur constitution hétérogène mettant en jeu des éléments de logique, d'électronique, d'électromagnétique et de mécanique;

- D'autre part, par le fait que ces unités dépendent fonctionnellement des organes centraux du système qui gèrent leur activation.

On donnera une classification des tests existants selon divers critères tels que : exhaustivité, fréquence d'emploi, mode d'exploitation... On définira les concepts permettant l'intégration de ces tests sous système d'exploitation et qui seront illustrés par l'exemple d'une réalisation faite dans le cadre de ce travail : COSU, programme COnversationnal générateur des tests Sous-Utilisateur, destiné à la vérification des périphériques des systèmes CII-10070 et IRIS 80.

Le problème de la vérification des dossiers fonctionnels en phase de mise au point du prototype et en intégration série, bien qu'il se pose avec acuité, n'a pas été formalisé jusqu'ici. Dans le chapitre V, on propose un

modèle de description des fonctions du système facilitant la compréhension des liens existant entre une description algorithmique de ces fonctions et le matériel qui les implémente. Puis à partir de ce modèle, on définira une méthode systématique d'élaboration de tests.

Dans le chapitre IV, on s'intéressera particulièrement aux tests d'acceptation et à l'outil logiciel nécessaire à leur mise en oeuvre.

Le dernier chapitre sera consacré au problème de la validation fonctionnelle au cours de la conception des systèmes. On rappellera brièvement la méthodologie de conception descendante et on justifiera la nécessité d'une représentation fonctionnelle multiniveaux correspondant à la progressivité de la démarche conceptuelle. On montrera l'intérêt de l'utilisation des réseaux de PETRI durant cette démarche et on présentera le système MAS qui est un outil de modélisation et de simulation respectant les principes énoncés.

CHAPITRE I

GÉNÉRALITÉS SUR LES TESTS

I - **TERMINOLOGIE** : FAUTE, PANNE et ERREUR |A1||A2||A3|

On rencontre souvent les mots défaut , erreur, panne, faute,... utilisés comme synonymes pour désigner une anomalie à l'origine d'une déviation du comportement attendu d'un système. Dans la suite de ce document, on utilisera chacun de ces termes dans un sens précis :

. Le terme *faute* sera réservé pour désigner une imperfection de conception. Une faute peut être dûe à une faille de raisonnement ou à un non respect des règles technologiques. On parlera de faute de nature logique et de faute de nature technologique ou physique.

. Le terme *défaut* sera utilisé pour désigner une défectuosité du matériel dûe à une imperfection de fabrication ou à l'usure. Le terme *panne* sera utilisé plus précisément pour indiquer l'effet produit par un défaut.

. Le terme *erreur* sera utilisé librement pour désigner un résultat incorrect fourni par le système pendant l'exécution d'une tâche donnée. L'apparition d'une erreur n'est pas nécessairement liée à la présence d'une faute ou d'un défaut. Outre les erreurs de programmation, on peut citer celles dûes à la surcharge instantanée d'un module du système; on peut penser par exemple aux erreurs de rythme dans une opération d'entrées/sorties.

La détection des fautes est du ressort des tests de validation des dossiers fonctionnels et matériels effectués lors de la mise au point du prototype. Plusieurs tests, effectués au cours de la conception et de la réalisation (validation fonctionnelle par simulation, test de qualification de la réalisation technologique), auront permis de prévenir au maximum de telles anomalies.

Les tests de production et de maintenance concernent la détection des défauts en fabrication série et le long de la vie des systèmes. Ces tests procèdent par identification du comportement des éléments constituant le système à la description qui leur est donnée dans les dossiers matériels. A la limite, le test de chaque élément peut être effectué isolément, sans tenir compte du rôle qu'il remplit dans la réalisation des fonctions globales du système.

Cependant, les tests de validation fonctionnelle ne sont jamais exhaustifs pour des systèmes complexes et il arrive que des fautes logiques ne se révèlent que trop tard à un stade déjà avancé de l'exploitation. Il arrive aussi que des fautes technologiques échappent aux contrôles de qualification et sont à l'origine d'erreurs ou de détériorations fréquentes qu'il est vain de réparer tant que ces fautes n'auront pas été cernées et corrigées.

II - BESOINS DE TEST EN PRODUCTION ET EN MAINTENANCE

II - 1. LES TESTS EN PRODUCTION

Pour des raisons pratiques (accessibilité) et afin de limiter les pertes en valeur ajoutée lors de la mise au point, les tests en production sont effectués à différents stades, et sévèrisés en amont |A4|.

1) Contrôle d'entrée des composants : (composants discrets et intégrés, généralement fournis par des fabricants spécialisés). Leur but est de contrôler la qualité des composants avant leur utilisation dans le montage des cartes; ils doivent garantir un taux inférieur à τ composants défectueux par carte équipée (τ généralement de l'ordre de 5/100).

2) Tests des cartes, fond de paniers, câbles, connecteurs, ... qui constituent ce que l'on peut appeler par *produit industriel élémentaire*.

3) Tests de sous-ensembles (cartes assemblées dans un ou plusieurs paniers). On trouve aussi à ce niveau les alimentations, pupitres de commande et de reconfiguration, sous-systèmes périphériques...

4) Tests au niveau système :

- *Tests d'acceptation*, on trouve essentiellement des tests de vérification fonctionnelle au niveau code. (Déverminage, mise au point en intégration).
- *Les diagnostics* qui servent à la fois au dépannage en mise au point et en maintenance.

II- 1.1. Les tests de contrôle d'entrée des circuits intégrés :

Il s'agit, le plus souvent, de méthodes adaptatives qui tiennent compte de la marge de confiance accordée au fournisseur et aussi de la complexité (SSI, MSI, Mémoire, LSI,...) et de la technologie utilisée (TTL, TTLS, ECL, MOS,...) |A5||A6||A7||A8|.

On distingue à ce niveau trois types de tests :

1) Les tests fonctionnels : ils ont pour but de s'assurer que les composants sont capables d'un fonctionnement correct dans des conditions d'environnement comparables à celles de leur utilisation (tension d'alimentation, températures, humidité relative,...).

2) Les tests paramétriques statiques : il s'agit d'un ensemble de tests marginaux qui permettent de comparer les limites réelles des caractéristiques statiques du composant (sensibilité aux variations des tensions et courants d'alimentation, conditions de charge, courants de fuite,...) à celles spécifiées par le fournisseur.

Suivant les fournisseurs, il est possible de trouver derrière un même schéma logique plusieurs techniques de réalisation; il faut descendre au niveau du schéma technologique (transistor, diode...) pour élaborer le test |A4||A7|.

3) Les tests paramétriques dynamiques : ils concernent les mesures des temps de propagation des signaux, la largeur des fronts de transition et les temps d'échantillonnage (*t-set up et t-hold*).

Dans une phase préliminaire à la fabrication série, l'exécution des trois types de test permet de déduire au vu des résultats obtenus :

a) La *qualification du composant* qui précise les règles d'utilisation dans les montages des circuits :
   - longueur des lignes d'attaque des circuits,
   - largeurs limites des impulsions d'horloge (t-set up et t-hold),
   - règles d'assemblage,
   - immunité aux bruits et phénomènes parasites de façon générale.

b) La *qualification du fournisseur* ou la marge de confiance à lui accorder. Les contrôles d'entrée ne peuvent être effectués sur la totalité des lots et suivant les trois types de test mais sur des échantillons prélevés. La fréquence des prélèvements et le nombre d'éléments échantillonnés à chaque fois seront déterminés durant cette

qualification. De même que les tests qui seront appliqués pour le contrôle en série, qui sont plus sensibles aux aspects critiques observés.

II- 1.2. Les tests de cartes :

Trois éléments de base constituent la carte :

. les circuits imprimés (cuivre),

. les circuits intégrés,

. les composants discrets.

Les tests sont tout d'abord effectués séparément pour chacun de ces éléments :

α) Nous avons vu en II-1.1. les divers types de tests effectués pour les circuits intégrés;

β) Des méthodes adaptatives sont aussi utilisées pour le test des composants discrets (prélèvement en fonction du lot et de la quantité de pièces dans le lot).

γ) Les tests des plaquettes nues portent essentiellement sur la continuité du conducteur et de l'isolation entre couches.

L'implantation des composants sur la plaquette puis la soudure créent un nouveau besoin de test, cette fois-ci sur plaquette équipée :

- détérioration éventuelle du composant (dûe à l'échauffement par exemple),
- court -circuits créés lors de la manipulation de la plaquette ou dûs à la présence de bavures,
- soudure imparfaite,
- erreur de montage.

On retrouve les trois types de tests :

1) Fonctionnels : test du circuit logique constituant la carte;

2) Paramétriques statiques : contrôle de la qualité des signaux sur variation des tensions et courants d'alimentation (continuité du conducteur, dispersion);

3) Paramétriques dynamiques : mesure des écarts par rapport aux temps de traversée nominals de la chaîne la plus longue et de la chaîne la plus courte.

Les tests électriques effectués sur plaquettes nues sont à nouveau appliqués pour les plaquettes équipées et pour les fonds de paniers, suivant des modalités particulières.

II - 1.3. Les tests de sous-ensembles :

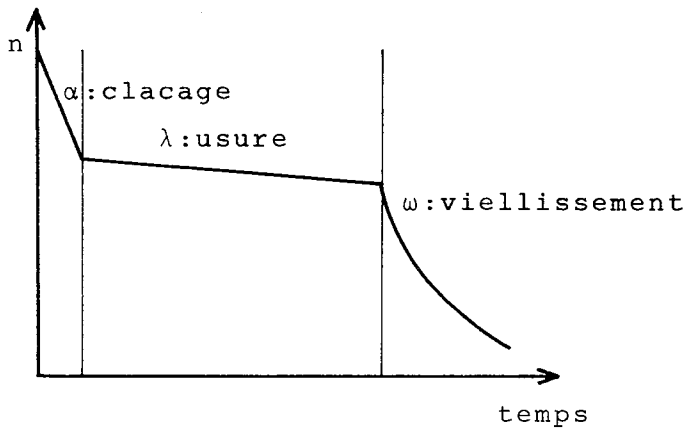
Par *sous-ensemble* on entend une partie délimitée du matériel qui constitue un module fonctionnel du système, c'est-à-dire, tel que son interface est complètement défini à un niveau relativement haut des spécifications fonctionnelles et matérielles : panier(s) de cartes constituant une unité mémoire ou un processeur quelconque, sous-système périphérique, pupitre de commande, ... etc.

Les tests à ce stade visent à effectuer la mise au point unitaire (locale) de sous-ensembles avant de passer au niveau d'assemblage final du système. Il s'agit essentiellement de tests fonctionnels appliqués à partir de l'interface standard de connexion des sous-ensembles.

II - 2. TESTS EN INTEGRATION DES SYSTEMES

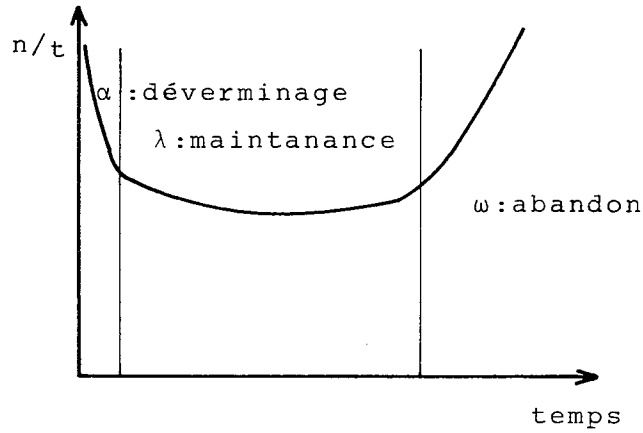
II - 2.1. Déverminage, usure et tests

Deux courbes aident à définir le rôle des tests dans la vie d'un système (figure I-1 et I-2) |A9|.



nombre de survies dans le temps pour des éléments identiques sans réparation

Figure I-1.



nombre de pannes par unité de temps

Figure I-2.

On distingue sur ces courbes trois phases α, λ et ω. On appelle *déverminage* (BURN-IN) la transition $\alpha \rightarrow \lambda$ et *usure* (WEAR-OUT) la transition $\lambda \rightarrow \omega$ |A10|.

Le rôle des tests en intégration (en particulier les tests *d'endurance* par bouclage intensif sur séquences de programmes) est d'accélérer la transition $\alpha \rightarrow \lambda$. Les opérations de maintenance visent à retarder la transition $\lambda \rightarrow \omega$. L'observation des histogrammes d'erreur permet de savoir dans quelle phase se trouve le système et donc de le situer par rapport aux courbes théoriques et moyennes.

II - 2.2. Tests d'acceptation - Validation fonctionnelle en intégration des systèmes

L'un des objectifs principaux des tests d'acceptation est donc d'effectuer la phase de déverminage afin de garantir au client une qualité de service satisfaisante. Il est bon que cette transition puisse s'effectuer dans des conditions d'environnement et de fonctionnement similaires à celles d'une exploitation réelle. Les tests d'acceptation permettent aussi d'ajuster certains réglages et de vérifier la bonne intégration de l'ensemble des modules.

Dans le cas où une configuration particulière est requise par le client, il faut s'assurer que celle-ci est capable de rendre les services attendus selon les normes garanties par les spécifications fonctionnelles du système. Bien que des calculs de performances auront été faits au préalable sur des modèles mathématiques ou par simulation, il convient de les vérifier sur le matériel et d'apporter, si besoin est, les rectifications nécessaires.

Ces tests sont repris en partie au moment de l'installation sur site ou en cas d'extension de la configuration : connexion de nouvelles unités d'échange ou de nouveaux dispositifs périphériques, passage d'un système monoprocesseur à un système biprocesseur, extension de la capacité mémoire.

Il convient pour ces tests de choisir les programmes qui permettent au mieux de recouvrir l'ensemble des situations de fonctionnement de simultanéité et de charges jugées critiques lors de l'exploitation du système.

II - 2.3. Les diagnostics

Il s'agit de programmes spécifiques à la vérification du matériel, qui permettent une détection et localisation rapides des défauts en cas de panne. Ce seront les mêmes tests qui seront utilisés pour le dépannage en maintenance.

II - 3. LES TESTS EN MAINTENANCE

L'action de maintenance vise l'entretien du matériel afin de préserver une qualité de service satisfaisante du système. Il y a lieu de planifier cette action pour qu'elle soit efficace et au moindre coût. Les caractéristiques de *maintenabilité* du système (accessibilité, reconfiguration, dispositifs de détection et d'enregistrement des contextes d'erreur...) contribuent à faciliter cette tâche |A11||A12||A13||A14||A15|.

On peut distinguer dans la maintenance deux types d'actions :

. Une action préventive :

a) Détecter si une panne existe avant qu'elle ne puisse altérer les résultats d'un traitement quelconque (*panne latente*). C'est le cas par exemple des tests effectués avant chaque chargement "à froid" du système d'exploitation (Early Morning Checkout), ou ceux appliqués de façon intermittente pour des organes où une probabilité de panne est grande ou bien là où les conséquences d'une erreur seraient catastrophiques. Les circuits de récupération d'erreur n'ayant jamais à intervenir lorsqu'il n'y a pas de pannes, il est bon de s'assurer périodiquement de leur fonctionnement.
Dans la mesure du possible, ces tests préventifs sont effectués parallèlement à l'exploitation afin de limiter les temps d'arrêt du système |A14|; on utilise les instants d'"oisiveté" des sous-ensembles du système particulièrement pendant les heures creuses de l'exploitation.
Ces tests sont repris systématiquement et de façon plus complète pendant les heures réservées à la maintenance périodique, exploitation arrêtée.

b) Entretenir un état "propre" de fonctionnement du système : tests de mesures et de réglages en cas d'usure éventuelle (essentiellement pour les sous-systèmes périphériques), contrôle des servitudes (alimentation, ventilation,...) et diverses opérations de nettoyage.

. Une action curative :

Dépannage après détection d'erreurs confirmant la présence de défauts. La procédure de test doit permettre de cerner le plus rapidement possible la panne afin d'effectuer les remplacements nécessaires et rétablir au plus vite

l'état de fonctionnement du système. Là aussi, il est important que les tests puissent s'effectuer autant que possible sans avoir à suspendre entièrement l'exploitation du système (isolation de l'unité présentant la panne et maintien au mieux d'une exploitation dégradée).

Un glissement des limites de bon fonctionnement du système (répétition fréquente d'erreurs intermittentes du même type) peut anoncer le vieillissement d'un composant ou une usure critique d'un organe mécanique ou électromagnétique. Des tests d'endurance (shake down tests) effectués dans des conditions de fonctionnement marginales (température, alimentation, code sévère ...) permettent d'accentuer l'effet produit par ces anomalies afin de pouvoir les localiser et les corriger.

III - TECHNIQUES DE TEST ET DE DIAGNOSTIC

III - 1. TEST DES CIRCUITS LOGIQUES

III - 1.1. Hypothèses et problèmes à résoudre

Les pannes dans les circuits logiques sont habituellement classées en deux types, les *pannes logiques* et les *pannes de propagation* |A3| :

. Une panne logique correspond à une modification de la fonction logique d'un composant élémentaire du circuit. Le modèle couramment utilisé est le *collage* (stuck-at) à l'entrée ou sortie du composant. Le collage à "0" ou à "1" d'une connexion signifie que le ou les défauts affectant le composant font qu'il se comporte comme si la connexion en question est effectivement bloquée à une tension basse (collage à 0) ou à une tension haute (collage à 1).

La génération d'un test de détection consiste alors à trouver un affichage d'entrées (ou une séquence d'affichages) tel que les résultats de sortie correspondants soient différents selon que la panne est présente ou absente. Dans cette procédure, on fait l'hypothèse que le circuit continue à avoir un fonctionnement logique même en présence de panne.

. Une panne de propagation correspond à un écart anormal dans les temps de propagation à travers le composant, induisant des impulsions parasites ou des transitions erronées (phénomènes d'aléas et de courses) |A46|.

La génération des tests pour des pannes de ce type nécessite une simulation temporelle fine du circuit |A15||A16| mais, généralement, les tests utilisés se basent plutôt sur des mesures et recettes expérimentales. Ce type de panne n'est pas considéré ici.

Les problèmes à résoudre sont les suivants :

1) Minimisation du nombre d'affichages pour le test global du circuit :

Le test d'un circuit peut être effectué à deux fins,

- La détection : il s'agit essentiellement de vérifier le bon fonctionnement; seule la connaissance de la présence ou de l'absence de défaut est suffisante. Les tests de ce type, qui sont en fait les plus fréquemment utilisés, doivent être les plus concis et les plus rapides possibles. On parle de test du type *go-nogo*.

- La localisation : quand un mauvais fonctionnement a été décelé, la séquence de test doit permettre de discerner l'élément constitutif du circuit qui en est la cause. Il faut trouver un compromis entre la longueur du test et le niveau du diagnostic souhaité.

2) Validité du test face à des pannes multiples :

Les tests basés sur l'hypothèse de la *panne unique* sont plus simples à concevoir et plus courts que ceux devant tenir compte des cas de *pannes multiples*. Mais si cette hypothèse est acceptable dans le cas des tests effectués en maintenance, se basant sur le fait que les pannes sont généralement détectées et corrigées dès qu'elles apparaissent, elle se justifie beaucoup moins pour les tests en production où la probabilité de pannes multiples est plus grande.

III - 1.2. Principe de génération des séquences de test

Les méthodes existantes pour la génération des séquences de test peuvent être classées selon deux critères principaux :

. Selon que la génération des séquences de test tient compte ou non de la réalisation du circuit. On parle de méthodes analytiques ou de méthodes fonctionnelles.

. Selon que le processus de génération de ces séquences procède d'un principe déterministe ou aléatoire |A3||A17||A18||A19||A20||A21|.

III -- 1.2.1. Méthodes analytiques

Il est possible de procéder de trois façons :

α) Par synthèse : pour une panne donnée, l'étude du schéma logique permet de déterminer quelles sont les séquences de détection correspondantes.

β) Par analyse : on applique au circuit une suite de combinaisons d'entrées puis on observe par simulation quelles sont les pannes qui sont détectables par cette séquence.

γ) Par synthèse et analyse : on combine les deux procédés. Dans une première phase, on détermine par synthèse une séquence de détection pour une panne donnée puis, dans une deuxième phase, on détermine par analyse quelles sont parmi les pannes qui restent celles qui sont détectables par cette séquence. Le processus est répété jusqu'à recouvrement de toutes les pannes considérées.

Les méthodes par synthèse ont l'avantage de pouvoir tenir compte des propriétés géométriques de la structure du circuit telles que la symétrie ou la répétitivité |A33||A34|. On parle de méthodes structurelles. En revanche, les méthodes par analyse sont plus facilement automatisables.

Méthodes par synthèse

On distingue deux méthodes principales :

α1) Méthode de la différence booléenne. La différence booléenne est définie comme étant le résultat du OU-exclusif entre la fonction logique représentant le comportement du circuit sain et la fonction logique représentant le comportement du circuit en présence de défaut. Si la différence booléenne n'est pas constamment nulle alors l'obtention d'une séquence de test est possible et est immédiate.

Cette méthode permet d'obtenir tous les tests possibles pour une panne donnée mais ne s'applique efficacement qu'aux circuits combinatoires de taille restreinte |A22||A23||A24|.

α2) Méthode du D-algorithme |A25|. Cette méthode se rattache au principe de la "sensibilisation" des chemins de propagation qui repose sur deux opérations de base :

a) Propager l'effet de la panne selon un ou plusieurs chemins jusqu'aux sorties primaires du circuit. C'est l'opération dite de *propagation*.

b) Remonter ces chemins jusqu'aux entrées primaires du circuit en cherchant les valeurs logiques qui permettent de manifester la panne et de sensibiliser les chemins de propagation choisis. C'est l'opération dite de *consistance*.

Ce principe, dû à ARMSTRONG |A26|, a été initialement appliqué à la sensibilisation unidimensionnelle (un seul chemin est examiné à la fois). Il a été constaté par la suite qu'il existe des circuits où des pannes détectables ne peuvent être propagées sur un seul chemin (problème des chemins qui convergent |A27|). La méthode du D-algorithme développée alors par ROTH |A25| permet grâce à l'artifice du *D-cube* d'effectuer la propagation en parallèle sur tous les chemins possibles.

Cette méthode est aujourd'hui appliquée tant aux circuits purement combinatoires, qu'aux circuits séquentiels synchrones et asynchrones |A28||A29| |A30||A31||A32|.

Méthodes par analyse

Deux mises en oeuvre prévalent :

β1) La technique de la simulation parallèle : pour une séquence donnée d'entrées, un circuit sain ainsi que N circuits obtenus par l'introduction de N pannes simples sont simulées en parallèle; la liste des pannes détectables par cette séquence est élaborée à partir des comparaisons des sorties obtenues |A35||A36||A37|. La méthode certes la plus utilisée est celle basée sur la génération aléatoire des séquences d'entrées (méthode de monte-carlo).

β2) La technique de la simulation déductive : seul le circuit exempt de défauts est simulé pour la séquence d'entrées; les résultats intermédiaires de cette simulation sont utilisés pour déterminer l'ensemble des défauts détectables par cette séquence. Cette technique a l'avantage d'éviter de simuler systématiquement le circuit pour chaque configuration de panne |A32| |A38||A39|.

III - 1.2.2. Méthodes fonctionnelles

Il s'agit de méthodes basées sur l'expérimentation du circuit testé et pour lesquelles aucune considération relevant de la structure du matériel réalisant le circuit n'est prise en compte.

α) Approche fonctionnelle exhaustive : elle consiste à vérifier la fonction du circuit en essayant toutes les combinaisons d'entrées possibles. Si N est le nombre d'entrées du circuit testé, il faut effectuer 2^N affichages; cette méthode n'est par conséquent envisageable que pour les circuits dont les entrées sont en nombre réduit.

Contrairement à ce qu'il vient en premier à l'esprit, le test d'un circuit combinatoire selon l'approche fonctionnelle exhaustive ne permet pas d'affirmer que toute panne logique permanente a pu être détectée. L'effet d'un défaut peut transformer un circuit combinatoire en un circuit séquentiel auquel cas seul un séquencement particulier des combinaisons d'entrées permet de manifester la panne |A40|.

β) Approches par identification d'automates : il s'agit, pour les cas des circuits séquentiels de taille réduite, d'effectuer une vérification du tableau d'état. Le problème est alors de trouver une séquence d'entrées E minimale telle que : si S est la séquence des sorties correspondantes de l'automate testé et S' celle de tout autre automate différent alors S et S' sont nécessairement différentes.

Il existe des méthodes qui donnent une solution à ce problème sous les hypothèses suivantes |A41||A42|.

1) L'automate testé est réduit,

2) Le nombre d'états de cet automate est (inférieur ou) égal au nombre d'états de l'automate de référence auquel il est comparé,

3) L'automate de référence admet une séquence de distinction ("distinguishing sequence").

La procédure d'identification consiste alors à positionner l'automate dans un état initial déterminé puis à vérifier toutes les transitions possibles entre états.

D'autres méthodes plus récentes (|A20|, Chap.III) permettent de substituer à l'hypothèse 3) des contraintes moins sévères mais conduisent, d'une façon générale, à des séquences beaucoup plus longues.

γ) Approches fonctionnelles aléatoires : la méthode de monte-carlo citée en III-1.2.1 peut être appliquée directement sur le circuit réel en simulant physiquement les défauts testés; les séquences de détection obtenues permettent d'établir un dictionnaire de pannes |A48|. On parle de méthode "par apprentissage".

L'approche aléatoire pûre consiste à générer les séquences d'entrées au moment du test effectif. Ces séquences sont appliquées en parallèle aux entrées du circuit testé et à celles d'un circuit réputé correct et les sorties correspondantes sont comparées; en cas de divergence, le circuit testé est déclaré défectueux. Le problème est alors de déterminer la longueur de la séquence à générer pour pouvoir assurer que toute panne présente sera détectée avec une probabilité donnée |A43||A44||A45|.

III - 2. TEST DE GRANDS ENSEMBLES

III - 2.1. Découpage et composition

Le test des grands ensembles (niveau sous-système ou système) est compliqué par le volume de logique qu'il faut analyser afin de bien saisir les liens existant entre les fonctions réalisées par ces ensembles et les éléments matériels qui les implémentent. D'un autre côté, les méthodes de génération des tests présentées plus haut, même automatisées, ne peuvent être utilisées de façon performante (efficacité, rapidité, coût) que pour des ensembles de taille relativement réduite.

Aussi est-on obligé d'avoir recours à un découpage en modules qui soient de taille plus abordable. La génération des tests s'effectue alors en deux étapes :

. Dans une première étape, on génère des séquences de test au niveau de chaque module;

. Dans une deuxième étape, on *compose* ces séquences au niveau de l'interface de l'ensemble global; autrement dit, on cherche les activations de l'ensemble qui permettent d'engendrer les séquences de test appropriées à l'entrée de chaque module et de propager les sorties correspondantes vers des points où leur analyse pourrait être effectuée.

La génération des tests au niveau de chaque module doit tenir compte à l'avance du problème de la composition : certaines séquences de test peuvent être impossibles à obtenir dans le fonctionnement normal du module au sein de l'ensemble global. De même, il n'est pas toujours possible de propager n'importe quelle séquence de sorties sans que l'information qui y est contenue ne soit altérée |A47||A48|.

La facilité avec laquelle on peut résoudre les contraintes de la composition ainsi que l'efficacité du test obtenu dépendent très étroitement du type de découpage considéré. On peut distinguer deux orientations principales régissant ce découpage :

. Un découpage *fonctionnel* qui facilite la procédure de composition. Chaque module est caractérisé par un ensemble d'opérations (multiplexage, décodage, calcul, mémorisation...) qui peuvent être facilement situées dans les phases d'exécution des fonctions globales.

. Un découpage *structurel* qui dégage les propriétés telles que la symétrie, la répétitivité, la régularité... |A33||A34| de nature à faciliter et optimiser la recherche des tests tant au niveau des modules qu'au niveau de la composition pour l'ensemble global.

Un découpage optimum prend en compte ces deux orientations. Cependant, ceci n'est pas toujours possible lorsque le système ou l'ensemble a été conçu sans tenir compte du test. Il faut alors trouver un compromis au détriment de l'exhaustivité du test ou de sa longueur. Quelquefois, des modifications simples, comme par exemple l'introduction des points de tests, suffisent à améliorer de façon sensible les performances des tests.

Des contraintes supplémentaires apparaissent lorsque ce découpage est à la base même d'une stratégie de test.

III - 2.2. Stratégies d'élaboration des tests pour le diagnostic des grands ensembles

On peut distinguer trois stratégies principales |A2| :

1) L'approche du type "Start-small" connue aussi sous le nom de "boule de neige". Dans cette approche, un minimum de matériel (le hardcore) est testé en premier puis chaque test supplémentaire prend en compte un nouvel incrément de circuiterie.

Cette approche présente deux avantages appréciables :

- La localisation des pannes est immédiate si l'on fait l'hypothèse que, quand une panne est détectée elle ne peut se trouver que dans l'un des circuits nouvellement testés.

- Elle permet de résoudre, dans une certaine mesure, le problème des pannes multiples : chaque panne détectée doit être réparée avant la poursuite du test et le découpage doit être suffisamment fin. De ce fait, elle s'adapte convenablement aux diagnostics de mise au point des sous-ensembles et systèmes en production.

La difficulté que pose l'élaboration du test selon cette approche réside essentiellement dans la recherche d'un découpage adéquat du matériel et d'un ordonnancement des circuits résultants pour leur test |A47|.

2) L'approche "Start-Big" : à l'opposé de l'approche "boule de neige", le test ici attaque d'emblée une grande portion de logique. Il s'agit essentiellement d'un test de détection optimisé. En cas de succès, le test se poursuit avec une autre grande portion. En cas de détection de défaut, il y a déroutement vers des tests de plus en plus fins jusqu'à cerner convenablement la panne.

Cette stratégie apparaît optimale pour les tests systématiques effectués au titre de la maintenance préventive. Mais l'élaboration du test est plus complexe |A48|. De plus, cette approche est inapplicable en pratique dans l'hypothèse des pannes multiples.

3) L'approche du type "Multiple Clue" : il s'agit d'une approche qui procède par recoupement d'une série de tests appliqués à l'ensemble du matériel. A la différence des deux premières approches, la détection d'un défaut n'arrête pas, ni modifie, le séquencement des tests. Chaque détection entraîne simplement le prélèvement du contexte; puis, quand tous les tests auront été effectués, l'analyse des contextes prélevés permet d'en déduire un diagnostic. (L'analyse de chaque contexte permet de calculer une "clé"; l'ensemble des clés obtenues sert ensuite à calculer une référence d'accès dans un dictionnaire de pannes).

L'avantage de cette approche est qu'elle n'exige pas une répartition stricte du matériel ni un ordonnancement précis des tests. En revanche, les algorithmes de diagnostic sont plus sophistiqués et deviennent pratiquement inextricables sous l'hypothèse des pannes multiples. De plus, la longueur de la séquence de test, qui peut parfois être grande, est constante, quel que soit le moment de la détection du défaut dans le test.

III - 3. MOYENS DE MISE EN OEUVRE DES TESTS - TENDANCES ACTUELLES

Les tests en production, du moins jusqu'au niveau sous-ensemble, s'effectuent sur des testeurs spécialisés |A49||A50||A51||A9|.

. La qualification des composants s'effectue sur des montages de référence très fiables (ce que les industriels appellent les JIGs). Ces montages sont utilisés, par la suite, pour calibrer les appareils industriels classiques sur lesquels seront effectués les tests de contrôle d'entrée en série.

. Les tests fonctionnels au niveau carte sont généralement du type *go-no-go*. Sur certains testeurs, l'utilisation de palpeurs mobiles permet l'accès (pour prélèvement) à tout point de jonction des composants sur la carte et fournit un moyen immédiat de localisation des défauts.

. Les tests au niveau sous-ensemble sont effectués sur des *Simulateurs d'Interfaces Fonctionnels* (SIF) pilotés par un processeur de test. Par l'intermédiaire du SIF, on peut simuler toute séquence de dialogue à l'interface du sous-ensemble testé et donc appliquer toute séquence de test.

. Enfin, les "diagnostics" comme tous les autres tests de maintenance sont appliqués à partir des moyens propres du système. Les séquences de test sont programmées au niveau d'un interface de commande standard (logiciel-matériel ou micrologiciel-matériel) et propagées jusqu'aux entrées des circuits concernés par le test.

Deux tendances principales apparaissent dans la prise en compte des besoins de test durant la conception du système :

- La première consiste à doter chaque sous-système ayant un fonctionnement autonome (unités de calcul, processeurs d'entrées/sorties, unités de contrôle de périphériques...) de moyens propres qui lui permettent de se tester par lui-même; c'est l'approche dite de l'autotest. Une procédure consiste à utiliser des microprogrammes de test qui sont, soit introduits en mémoire de commande du sous-système par l'intermédiaire d'une voie spéciale de maintenance, soit figés dans des cartes mémoire morte que l'on substitue à la mémoire de commande du sous-système au moment du test.

- La deuxième consiste à disposer pour chaque sous-système d'une liaison spéciale qui permet d'effectuer entièrement son test à partir d'un processeur externe servant de console de maintenance. L'une des méthodes concernant l'implantation d'une telle liaison est celle du DSR "Diagnostic Shift Register" chez XDS|A52| connue aussi sous le nom de MICROBUS chez CII |A53|.

Principe : |A54|

Un sous-ensemble logique peut être décomposé en deux parties disjointes :

α) Les éléments de mémorisation (bascules, registres...);

β) La logique combinatoire.

- On peut effectuer le diagnostic des éléments de mémorisation séparément si on a pu fournir un accès supplémentaire à chaque bascule.

- La partie combinatoire, elle, peut être décomposée en réseaux logiques bornés par des registres d'entrée et de sortie. Les fonctions de chaque réseau peuvent être testées en forçant des états dans les registres d'entrée puis en lançant une ou plusieurs impulsions d'horloge. La lecture des registres de sortie permet l'observation des fonctions.

L'idée consiste alors à utiliser les sorties parallèles des registres pour le fonctionnement normal du système et d'établir une liaison série entre ces bascules de façon à former un seul grand registre à décalage : le DSR (Figure I-3). Les extrémités du DSR sont conduites, grâce à une liaison spéciale, jusqu'au processeur de test.

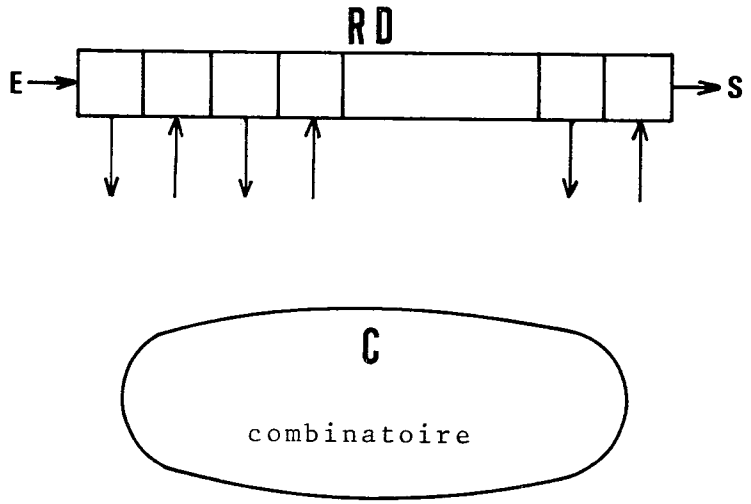


Figure I-3.

Cette liaison spéciale, que l'on peut appeler "Bus de Maintenance" sera constituée :

i) D'un bus de commande de diagnostic,

ii) D'un bus d'entrée de données,

iii) D'un bus de sortie de données.

Les DSR, une fois testés (par une procédure relativement simple) sont utilisés comme points de test pour la partie combinatoire.

On peut citer les avantages suivants :

1) La procédure d'application du test est uniforme et systématique aussi bien au niveau système qu'au niveau sous-ensemble.

2) Dans les deux cas, le test des réseaux combinatoires est effectué à la cadence normale de l'horloge.

3) L'élaboration des séquences de test est la même au niveau sous-ensemble ou au niveau système.

4) Le concepteur à une connaissance à priori de la logique à ajouter aux fins du test et du diagnostic.

5) Il n'y a pas de contraintes d'ordonnancement entre les réseaux combinatoires ainsi isolés, et la taille de ces réseaux permet la génération des vecteurs de test selon des méthodes maitrisées.

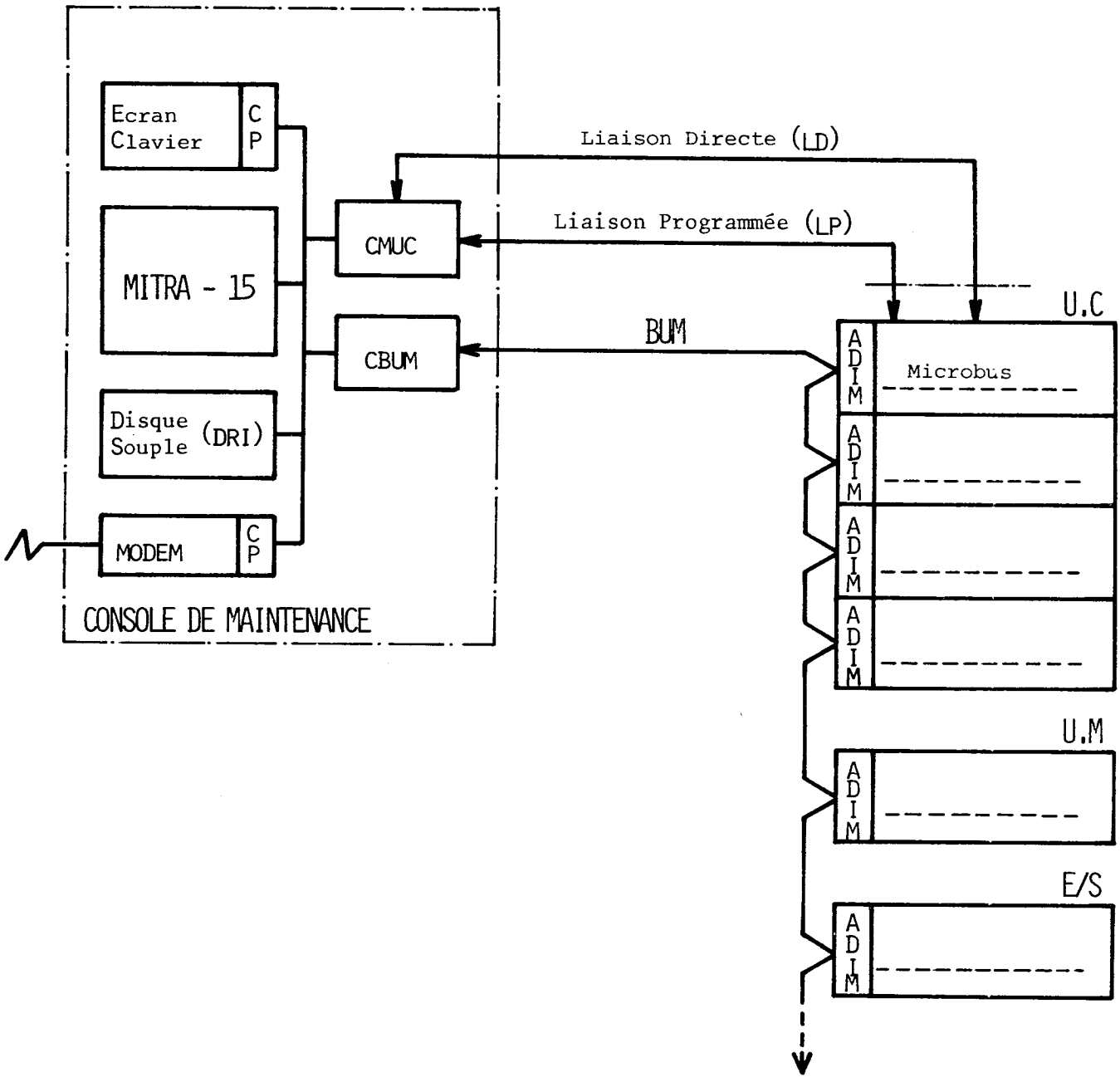
On note, en revanche, quelques inconvénients :

i) Le DSR introduit des éléments logiques non-fonctionnels. Ceci a comme conséquence un surplus de coût et des possibilités de perturbation du fonctionnement normal du système, dûes aux pannes dans la logique ainsi ajoutée.

ii) Les temps de test peuvent être relativement lents (lecture et écriture des registres par décalages).

iii) Dans le cas où plusieurs technologies sont utilisées dans la réalisation du sous-ensemble, des problèmes d'incompatibilité peuvent apparaître dans la liaison de toutes les bascules sur un même DSR.

Cette méthode a été largement exploitée par la CII pour la réalisation des machines de la nouvelle gamme UNIDATA 7740/60 (Figure I-4). Les retombées se situent au niveau du test des cartes où le même principe est appliqué.



BUM : Bus de Maintenance
ADIM : Adaptateur d'Interface de Maintenance
LD et LP Liaisons de dialogue en fonctionnement normal de l'UC

Figure I-4.

Liaisons Console de Maintenance pour le
Système UNIDATA 7760

CONTEXTE DU TEST				NATURE DES TESTS ET OUTILS MIS EN OEUVRE				APPROCHES D'ELABORATION DES TESTS		
Conception	Production	Maintenance		Types d'anomalies visées		Tests de vérification fonctionnelle	Tests de vérification matérielle	Start-Big	Start-small	Multiple-clue
X			Validation fonctionnelle en phase de conception	Fautes	Simulation sur modèles haut-niveaux (GPSS, SIMULA, SIMSCRIPT, APL...)	X				
X			Validation structurelle et technologique en phase de réalisation des prototypes	Fautes	Simulation au niveau structurel fin et au niveau circuit électrique (CASSANDRE, DDL, CDL, IMAG2,...,); expérimentation du matériel réalisé.	X	X			
X			Validation des dossiers fonctionnels et matériels des premiers prototypes	Fautes + Défauts	Vérification fonctionnelle au niveau code machine et diagnostics	X	X	Validation des dossiers fonctionnels	Mise au point	
	X		Tests de contrôle d'entrée des composants	Défauts	Tests fonctionnels, statiques et dynamiques effectués sur testeurs spécialisés		X			
	X		Tests au niveau cartes	Défauts	Tests fonctionnels, statiques et dynamiques effectués sur testeurs		X			
	X	(X)*	Tests au niveau sous-ensembles	Défauts	Vérification et mise au point unitaire du matériel effectuées sur des SIFs		X	En maintenance préventive	En production et en maintenance curative	En maintenance curative
	X		Tests au niveau système en intégration et en maintenance	Fautes + Défauts	Tests d'acceptance au niveau code machine	X		Tests d'acceptance et en		
	X	X			Tests de diagnostic (code ou micro-code)		X	maintenance préventive	tests de diagnostics	En maintenance curative

* Quand les sous-ensembles sont dotés des moyens d'autotest ou d'une liaison spéciale de maintenance.

CHAPITRE II

PROBLEME DU TEST DES MODULES D'INTERFACE

I - INTRODUCTION

On se situe dans le cas d'une décomposition d'un système au niveau de sous-systèmes fonctionnant en asynchrone. On distingue pour chaque sous-système deux parties de logique :

- Une partie S qui constitue la logique d'interface du sous-système (circuits synchroniseurs, arbitres,multiplexeurs, etc...).

- Une partie P qui réalise les fonctions proprement dites du sous-système.

On présentera dans ce chapitre le problème du test des circuits des parties S. On rappellera la fonction de quelques circuits types et les problèmes qui se posent lors de leur conception et on donnera alors les définitions et les justifications des différents tests nécessaires. L'accent sera mis essentiellement sur le problème de l'activation de ces tests au niveau de commande du système global.

II - CIRCUITS TYPES DANS LES MODULES D'INTERFACE

On peut distinguer essentiellement trois circuits types :

Les circuits synchroniseurs

La fonction d'un circuit synchroniseur pour un sous-système est de mémoriser puis cadrer un signal d'entrée sur les impulsions de l'horloge du sous-système. La solution conventionnelle consiste à utiliser des éléments bistables en prenant soin de garantir que les sorties des bistables atteignent un état stable logiquement défini et dans un temps suffisamment court chaque fois qu'un signal arrive à l'entrée du sous-système.

Un signal d'entrée peut arriver avant, pendant ou en même temps qu'une impulsion d'horloge. CHANEY et MOLNAR |B1| ont montré que certains des bistables les plus utilisés tels que MOTOROLA MC1016 (ECL), SN7410 et SN7474 (TTL), certains recouvrements des transitions d'entrée et des transitions d'horloge porvoquent des oscillations aux sorties des bistables. Une circuiterie protectrice additionnelle doit être prévue afin de prévenir de telles situations qui sont susceptibles, sinon, d'amener un fonctionnement imprévisible des échanges entre les sous-systèmes |B2|. CATT dans |B3| met en évidence le coût en temps d'exécution et en matériel que représente l'adjonction d'une circuiterie protectrice.

Les circuits arbitres

Un sous-système donné peut constituer une ressource particulière dont l'accès est partagé entre plusieurs sous-systèmes utilisateurs. Le rôle du circuit arbitre dans le module d'interface est de résoudre les conflits quand deux ou plusieurs utilisateurs demandent en même temps l'accès à cette ressource. Généralement, la résolution des conflits est effectuée selon des règles de priorité bien précises.

La solution la plus communément adaptée est de ce type : dans un premier temps, on fige la configuration des demandes présentes à l'entrée du circuit arbitre puis, dans un deuxième temps,on détermine l'attribution sur cette configuration stable. Il se pose le même problème de l'utilisation des éléments de synchronisation cité plus haut qui nécessite la prise en compte d'un certain nombre de précautions. Les incidents qu'il faut prévenir sont les suivants :

- Sélection par l'arbitre d'une demande fantôme (qui n'a pas été positionnée) ou inversion de l'ordre de priorité entre les demandes présentes.

- Sélection simultanée de plusieurs demandes.

- Génération d'aléas critiques sur les sorties du circuit arbitre.

On trouve dans la littérature des solutions qui utilisent des technologies spécialisées. PATIL dans (|B4|) utilise une logique à seuil (threshold not gates). PLUMMER dans |B5| utilise un "circuit de décision à fronts rapides" (Decide-one-shot). PEARCE, FIELD et LITTLE |B6| utilisent des lignes à retard. DAVID et DESCOTES-GENON, qui réalisent des arbitres asynchrones à partir de CUSA |B7|, utilisent une double inhibition par rebouclage des sorties internes des cellules gérant les sorties du circuit arbitre.

Les circuits de sélection des chemins de données

Ce sont les circuits utilisés pour le multiplexage de l'information quand plusieurs chemins d'entrée ou de sortie de données sont possibles. Il s'agit de circuits classiques (décodeurs, multiplexeurs,...) que l'on retrouve dans la plupart des réalisations logiques. Cependant, leur utilisation dans les modules d'interface leur confère une importance particulière. Une panne dans ces circuits peut amener une contradiction entre le positionnement d'un chemin de données et le choix effectué par le circuit arbitre; l'erreur qui en résulte sera d'autant plus grave qu'elle passera pratiquement inaperçue. Une redondance dans le codage des données et dans la réalisation des circuits de multiplexage est nécessaire afin de se prémunir contre de telles erreurs.

III - PROBLEME DU TEST DES CIRCUITS D'INTERFACE

III - 1. DEFINITIONS

III - 1.1. Tests statiques et tests dynamiques

On a déjà avancé, dans le chapitre précédent (§II-1.2) la distinction entre test statique et test dynamique sans en préciser les définitions correspondantes :

1) On dira qu'un test est *statique* s'il est effectué de la manière suivante : chaque fois qu'un vecteur d'entrée est appliqué, on attend que le circuit se stabilise avant de prélever les valeurs aux sorties et d'appliquer un nouveau vecteur. Il est sous-entendu que cette stabilisation doit être atteinte au bout d'un temps maximum t_m.

2) On dira qu'un test est *dynamique* si l'application des entrées de test, ainsi que le prélèvement des sorties correspondantes sont soumises à des contraintes temporelles précises, indépendamment de l'état de stabilisation du circuit testé.

3) On dira qu'une panne α crée une *plage temporelle θ de fonctionnement critique* (intervalle de temps θ) si l'effet de cette panne ne se manifeste que quand une séquence donnée d'évènements d'entrée se produit dans un intervalle de temps inférieur à θ.

III - 1.2. Tests unitaires et tests de simultaneité

On se place dans le cas du test au niveau système, c'est-à-dire pour lequel le test de chaque sous-système ne peut être effectué qu'"à travers" les sous-systèmes qui le commandent.

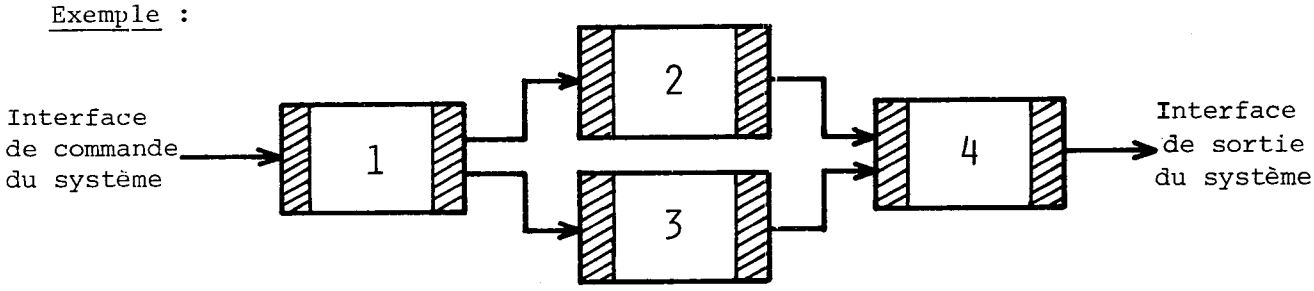
Un test pour un sous-système peut être alors de l'un des deux types suivants :

1) *Unitaire* si sa mise en oeuvre requiert l'activation d'un seul des sous-systèmes commandant directement le sous-système sous test.

Un sous-système S est dit commandant directement un sous-système S' s'il dispose d'une voie d'accès à l'interface immédiat de ce sous-système.

2) *De simultaneité* si sa mise en oeuvre nécessite l'activation simultanée de deux ou plusieurs sous-systèmes commandant directement le sous-système sous test.

Exemple :



- Le sous-système 1 commande les sous-systèmes 2, 3 et 4. Il commande directement les sous-systèmes 2 et 3.

- Les sous-systèmes 2 et 3 commandent directement le sous-système 4.

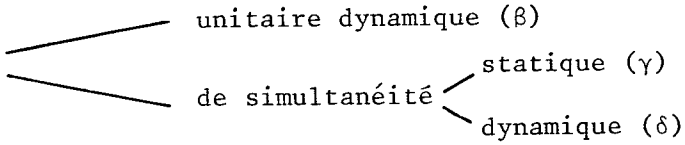
- Un test unitaire pour le sous-système 4 est un test qui utilise soit les sous-systèmes 1 et 2 soit les sous-systèmes 1 et 3.

- Un test de simultanéité est un test qui utilise obligatoirement les deux sous-systèmes 2 et 3 pour commander en simultanéité le sous-système 4.

III - 2. EXEMPLES DE PANNES DANS LES MODULES D'INTERFACE NECESSITANT DES TESTS DYNAMIQUES ET DES TESTS DE SIMULTANEITE

On distinguera 4 types de pannes :

- pannes pouvant être détectées par un test unitaire statique (α)
- panne ne pouvant être détectées que par un test



Pannes nécessitant un test de simultaneite statique

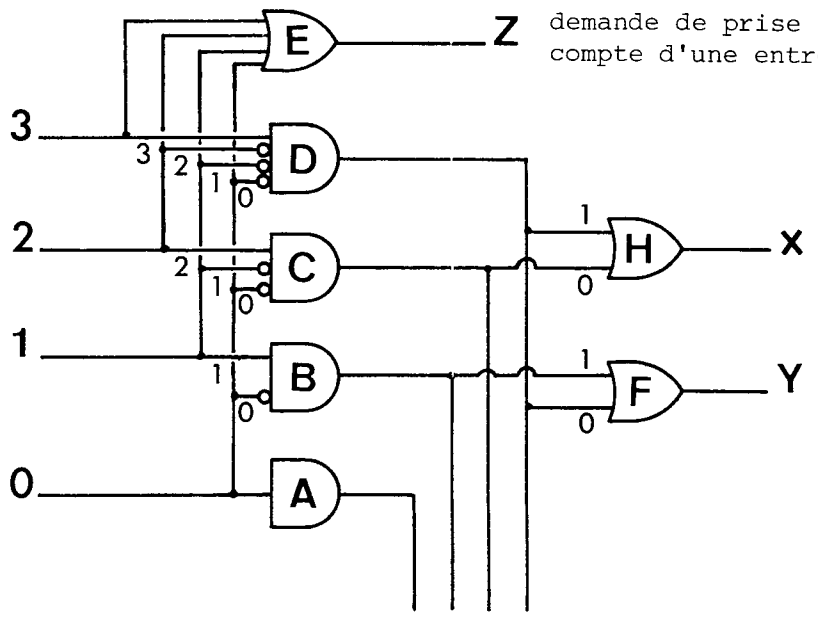
Les deux exemples qui suivent mettent en évidence deux types de manifestations fonctionnelles des pannes du type γ, à savoir :

- pour les circuits arbitres synchrones
  - . inhibition d'entrées prioritaires et/ou
  - . création d'entrées fantômes
- pour les circuits multiplexeurs
  - . sélection de fausses données

- Exemple 1 : Pannes dans un circuit de priorité

Quand toutes les demandes utilisateurs sont synchronisées à leur arrivée à l'interface du sous-système, le problème de l'arbitrage cité en II peut être résolu de manière simple par un circuit du type présenté en Figure II-1.a. Dans cet exemple, les entrées 0, 1, 2 et 3 correspondent aux demandes de connexion de service émanant de 4 sous-systèmes indépendants. La sortie Z informe de la présence d'une demande à l'entrée du sous-système. Les sorties des portes A, B, C et D sont directement utilisées pour mémoriser le numéro de l'entrée active qui vient d'être choisie; ce numéro est ensuite codé sur les deux sorties X et Y.

Un test unitaire implique qu'une seule des entrées 0, 1, 2 ou 3 soit positionnée à la fois. L'analyse du circuit montre que les pannes B_0^0, C_0^0, C_1^0, D_0^0, D_1^0 et D_2^0 ne sont pas détectables de manière unitaire et nécessitent donc la présence simultanée de deux ou plusieurs demandes à l'entrée du circuit. Ces résultats sont explicités dans le tableau de la Figure II-1.b. La notation $P_k^{0|1}(n_i \rightarrow n_j)$ est utilisée pour représenter un collage à 0 ou à 1 de l'entrée k de la porte P; n_i et n_j sont les numéros de l'entrée prioritaire qui sont donnés par le circuit respectivement en l'absence et en présence de la panne.



3 2 1 0	x y
0 0 0 1	0 0
0 0 1 0	0 1
0 1 0 0	1 0
1 0 0 0	1 1

Figure II-1.a

Pannes	Vecteurs de détection	x, y	Effet de la panne
B_0^0	$\overline{32}\overline{1}0$, $(\overline{3}\overline{2}10)$ $(3\overline{2}10)$, (3210)	0 1	$(0 \to 1)$: inhibe l'entrée prioritaire 0
C_0^0	$\overline{3}2\overline{1}0$, $(32\overline{1}0)$	1 0	$(0 \to 2)$: inhibe l'entrée prioritaire 0
C_1^0	$\overline{3}21\overline{0}$, $(321\overline{0})$	1 1	$(1 \to 3)$: inhibe 1 et crée fantôme 3
D_0^0	$3\overline{21}0$	1 1	$(0 \to 3)$: inhibe l'entrée prioritaire 0
D_1^0	$3\overline{2}1\overline{0}$	1 1	$(1 \to 3)$: inhibe l'entrée prioritaire 1
D_2^0	$32\overline{10}$	1 1	$(2 \to 3)$: inhibe l'entrée prioritaire 2

Figure II-1.b

- Exemple 2 : Pannes dans un circuit de sélection des chemins de données

Supposons que les sorties (x,y) du circuit de l'exemple précédent sont utilisées pour multiplexer les entrées de données à l'interface du sous-système. Si la largeur des données est n, on peut utiliser n multiplexeurs à 4 entrées pour réaliser le circuit multiplexeur requis à l'interface du sous-système (Figure II-2).

La figure II-3.a donne une réalisation possible d'un multiplexeur à 4 entrées. L'analyse du circuit montre que les pannes M_1^0, M_2^0, N_1^0, N_2^1, P_1^1, P_2^0, Q_1^1 et Q_2^1 ne peuvent être propagées qu'en imposant certaines valeurs de données sur les entrées associées à deux chemins de données différents; opération qui nécessite l'activation simultanée des deux sous-systèmes correspondants (voir tableau Figure II-3.b).

Pannes nécessitant un test de simultanéité dynamique

Entre l'instant où un circuit arbitre est appelé à décider du choix d'une demande parmi celles qui sont présentes et l'instant où cette décision devient effective, l'entrée de nouvelles demandes doit être inhibée pour permettre au circuit d'effectuer la sélection sur une configuration stable. Ce temps est relativement long quand il s'agit d'une sélection non-linéaire où le circuit arbitre doit mémoriser un historique des dernières demandes servies |B5|.

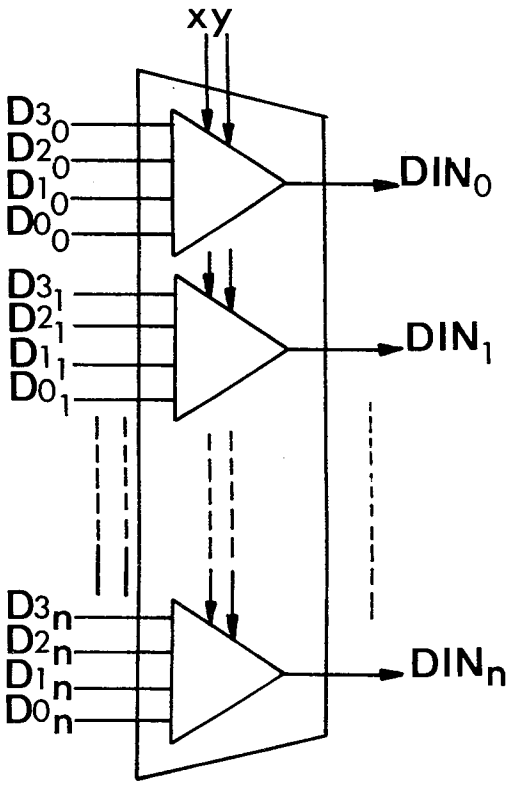


Figure II-2

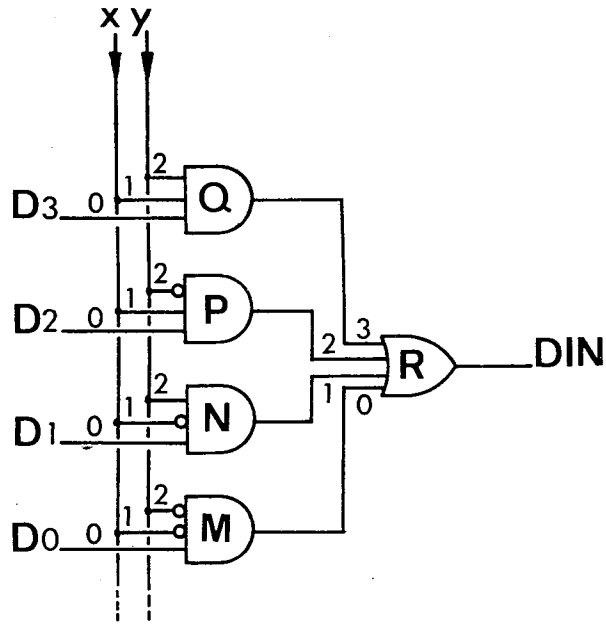


Figure II-3.a

Panne	Vecteur de détection					Effet
	D0	D1	D2	D3	x y	
M_1^0	1	x	0	x	1 0	(0 → 1)
M_2^0	1	0	x	x	0 1	(0 → 1)
N_1^0	0	1	x	0	1 1	(0 → 1)
N_2^1	0	1	x	x	0 0	(0 → 1)
P_1^1	0	x	1	x	0 0	(0 → 1)
P_2^0	x	x	1	0	1 1	(0 → 1)
Q_1^1	x	0	x	1	0 1	(0 → 1)
Q_2^1	x	x	0	1	1 0	(0 → 1)

Figure II-3.b

Nota : Les pannes M_1^0, M_2^0, N_1^0 et P_2^0 ne sont pas détectables dans l'hypothèse de la panne simple en raison de l'incompatibilité entre le positionnement des entrées D_i et des valeurs de x et y. Seul un test non-fonctionnel [B8] permet de les détecter.

Dans le cas d'un arbitre synchrone, ces deux instants correspondent à deux fronts successifs d'horloge. D'une façon générale pour les arbitres asynchrones, la place temporelle d'inhibition dépend uniquement des instants de mise à 1 et de remise à 0 des demandes et se situe donc de manière tout à fait quelconque par rapport au temps de référence de la ressource. Il peut exister des pannes dont la mise en évidence nécessite qu'un test dynamique de simultanéité soit appliqué dans les limites de cette plage d'inhibition. L'exemple suivant montre un tel cas de panne.

- Exemple 3 : Arbitre asynchrone de PLUMMER |B5|

Il s'agit d'un arbitre fonctionnant selon les conventions de dialogue Demande/Acquittement (Figure II-4.a). Une "logique de port" est utilisée à chaque entrée i du circuit arbitre pour mémoriser l'arrivée d'une demande utilisateur sur ce port ($R_i \rightarrow ACT_i$) et acheminer le signal d'acquittement correspondant ($ACK \rightarrow A_i$). Quand le circuit arbitre est amené à prendre une décision, il met à zéro un signal AWAIT qui inhibe l'entrée de nouvelles demandes, et active un circuit de priorité qui décide laquelle parmi les demandes présentes est prioritaire; il remet alors à zéro momentanément toutes les sorties ACT_j qui ne sont pas prioritaires ($PRI_j=0$). La ressource est allouée au port d'entrée ayant la sortie ACT maintenue à 1. En fin de traitement, le signal d'acquittement rendu par la ressource ($A_0 \rightarrow ACK$) est transmis à l'utilisateur à travers la logique de port correspondante. L'enchaînement de ces actions est illustré sur le chronogramme de la figure II-4.b.

La figure II-4.c donne le schéma détaillé de cet arbitre pour le cas de deux utilisateurs. Une priorité linéaire donnant avantage au port 1 peut être réalisée simplement en bouclant la sortie $\overline{ACT1}$ sur l'entrée $\overline{PRI2}$ de la logique de port associée au port 2. Pour des détails plus complets, le lecteur se reportera à la référence |B5|.

Une panne du type collage à 1 à l'entrée AWAIT des portes I1 ou I2 ne peut pas être détectée par un test unitaire. Le rôle, par exemple, de la remise à zéro de l'entrée AWAIT de I2 est :

1) D'inhiber l'entrée éventuelle d'une demande R2 si celle-ci n'est pas présente au moment où l'arbitre décide de prendre en compte une demande sur le port 1, ou

2) De permettre la remise à zéro momentanée de ACT2 si la demande R2 n'est pas prioritaire (R1 et R2 ont été demandées simultanément).

Dans le cas où seule R1 ou R2 est positinnée à la fois, le collage à 1 de l'entrée AWAIT de I2 n'a pas d'effet.

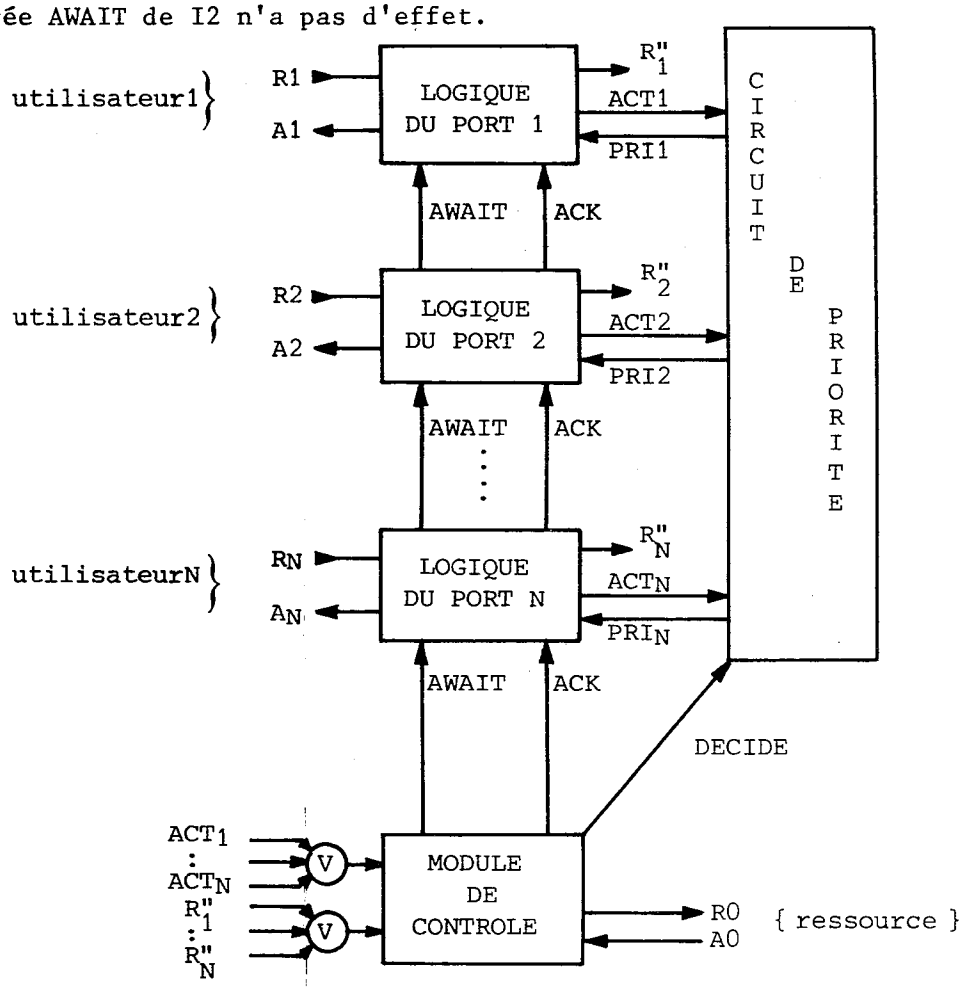
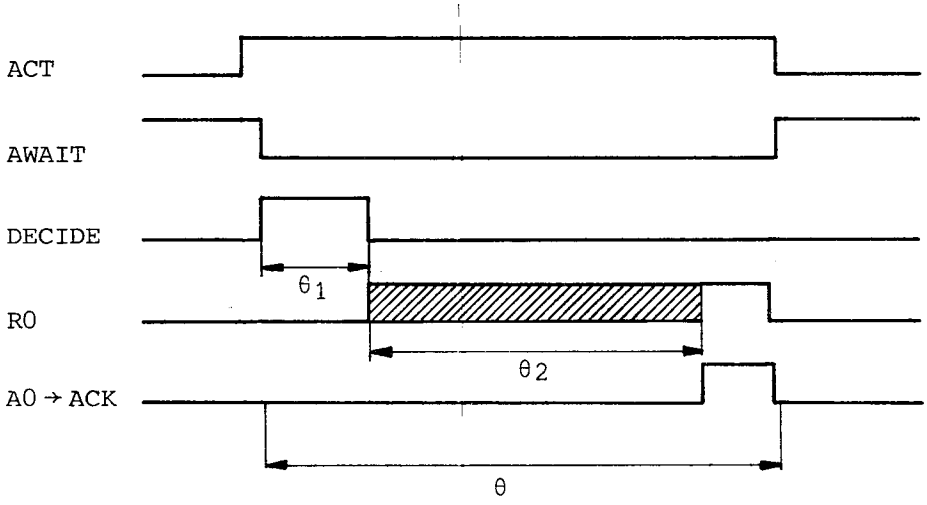


Figure II-4.a



θ_1 : stabilisation du circuit et décision de la demande la plus prioritaire (RAZ des demandes non prioritaires)

θ_2 : traitement effectué pour le compte de la demande élue

Figure II-4.b

Dans le cas où R1 et R2 sont positionnées en même temps, la panne considérée a pour effet l'acquittement de la demande R2 en même temps que la demande R1 sans que le traitement correspondant à R2 ne soit effectué (Figure II-5). Il s'agit là du cas d'une panne qui crée une plage temporelle de fonctionnement critique de largeur θ (l'effet de cette panne ne se manifeste que sur occurence des évènements R1 et R2 à une distance inférieure à θ; voir chronogramme Figure II-4.b). Cette panne peut être détectée par un test de simultanéité statique.

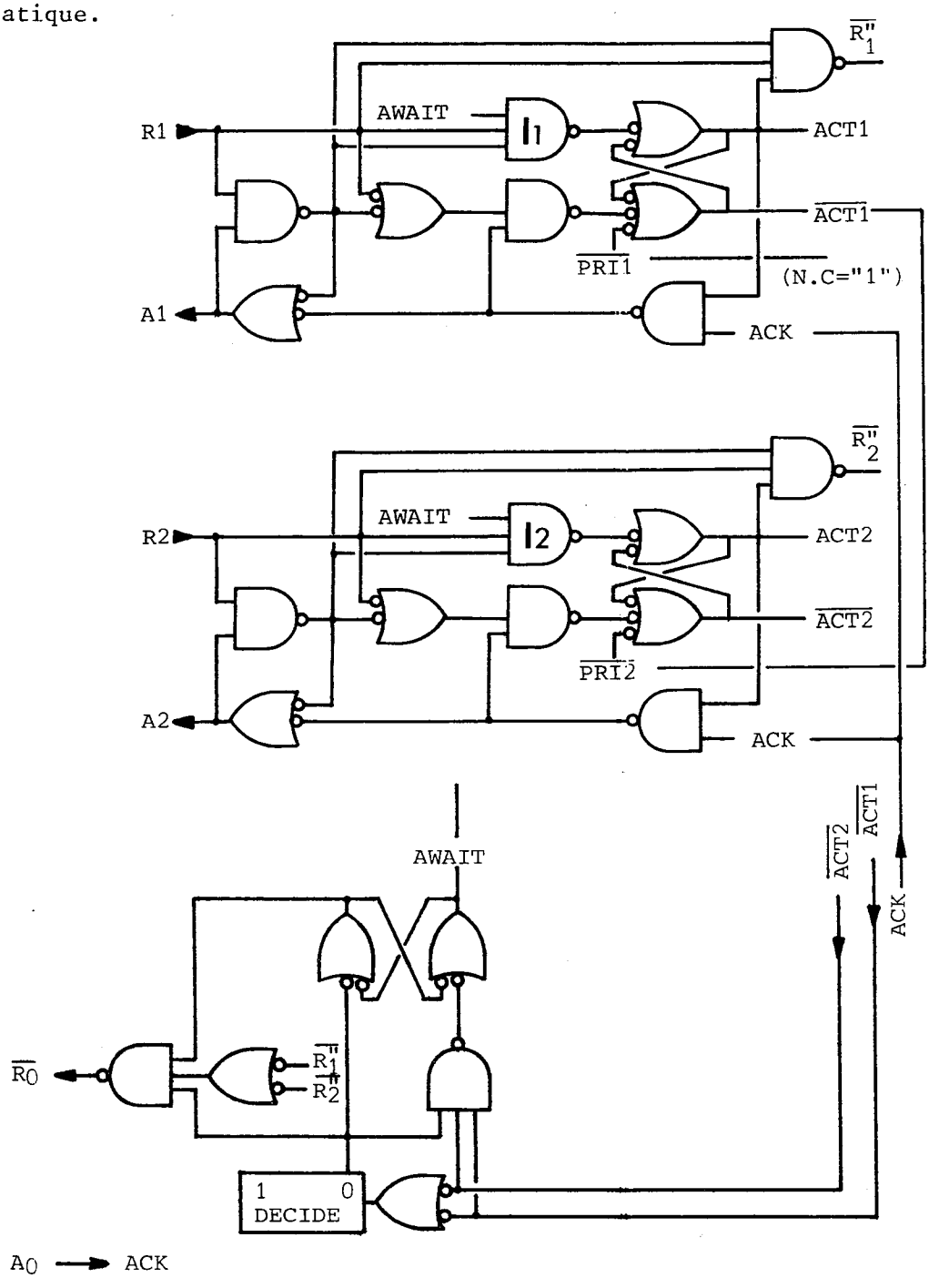


Figure II-4.c

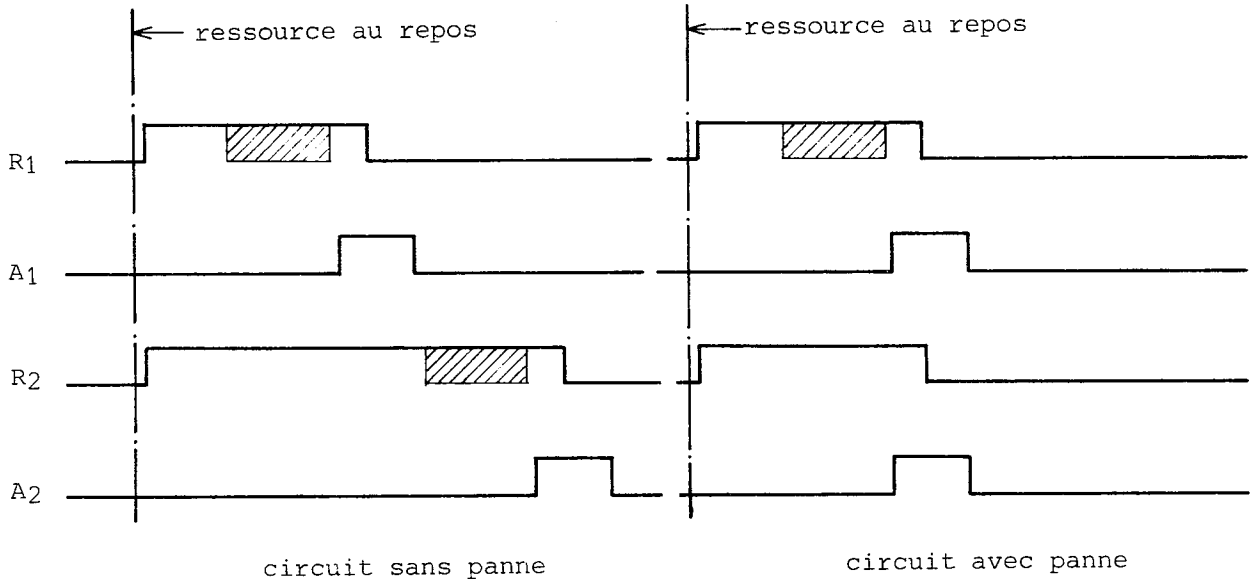


Figure II-5

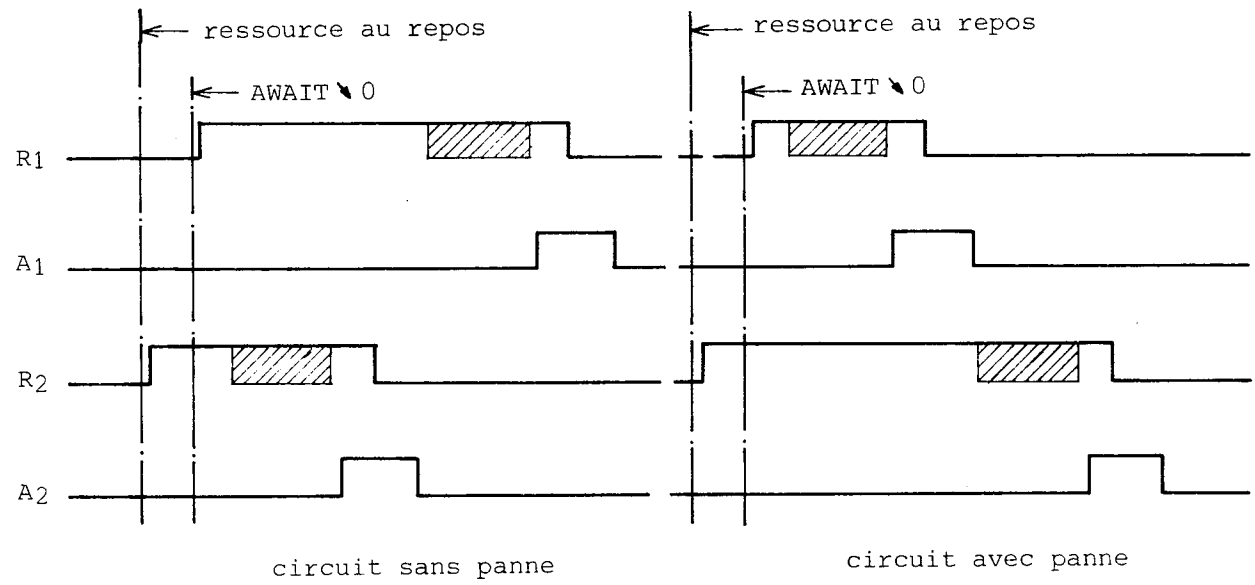


Figure II-6

Le cas du collage à 1 à l'entrée AWAIT de la porte I1 est plus difficile à traiter. La mise en évidence de cette panne nécessite le positionnement de R2 en premier puis, quand le signal DECIDE passe à "1", le positionnement de la demande R1. En cas de fonctionnement sans panne, la demande R2 est servie la première puis la demande R1 est servie à son tour (chronogramme en Figure II-6). Si la panne est présente, les conséquences sont imprévisibles puisque cela dépend de la manière dont réagit l'ensemble (arbitre + ressource) vis à vis d'une transition tardive de la sortie ACT1 le long de l'intervalle θ. Toutefois, si l'on suppose que le signal DECIDE est suffisamment large pour permettre une stabilisation satisfaisante du circuit, il suffirait, pour mettre en évidence la panne, d'appliquer l'entrée R1 aussitôt après la montée de DECIDE. La présence de la panne fera que la demande R1 passe en priorité alors qu'elle devrait être retardée jusqu'au tour suivant. Cette panne crée la même plage temporelle de fonctionnement critique que la panne précédente; mais sa détection nécessite un test de simultanéité dynamique.

IV - PROBLEME DE MISE EN OEUVRE DES TESTS

Les méthodes analytiques dont on dispose à l'heure actuelle s'appliquent convenablement à la génération des tests statiques et dynamiques localement au niveau de chaque circuit. Le problème se pose quand il s'agit de composer ces tests au niveau de l'interface de commande du système global.

La composition n'est théoriquement possible que pour le cas des tests unitaires statiques :

- La génération des tests dynamiques ne peut se faire que selon des méthodes descendantes (par analyse) moyennant une simulation temporelle fine des circuits |B9|. Il est techniquement impossible d'appliquer de telles méthodes à des systèmes de taille importante.

- La composition des tests de simultanéité se heurte de façon générale aux restrictions des moyens d'activation et de contrôle imposés au niveau de commande du système global. Des exemples de telles restrictions sont les suivants : considérons le cas de la configuration donnée en figure II-7, les moyens standards d'activation du système ne permettent pas d'amener de façon sûre une situation de simultanéité d'accès mémoire entre l'unité centrale et l'unité d'échange. De même que ces seuls moyens ne permettent pas d'amener de façon sûre des réponses simultanées des unités de contrôle des périphériques

vers l'unité d'échange. Pourtant, de telles situations sont très probables dans le fonctionnement normal du système.

On peut donner un dernier exemple qui illustre les difficultés de la composition des tests au niveau système pour les modules d'interface.

- Exemple 4 : Pannes dans un circuit synchroniseur

La figure II-8.a donne une réalisation possible d'un circuit synchroniseur |B10|. Les chronogrammes correspondant au fonctionnement du circuit sans panne sont donnés en figure II-8.b pour les différents cas possibles de recouvrement des impulsions d'horloge C et des impulsions d'entrée P. Une analyse du fonctionnement du circuit montre que le collage à 1 de l'entrée Y2 du NAND A1 crée une plage temporelle de fonctionnement critique pour un recouvrement du type (2) (Figure II-8.c). Le test de cette panne nécessite la maîtrise de la commande des deux signaux C et P, chose qui n'est pas possible par les seuls moyens d'activation au niveau de commande du système global.

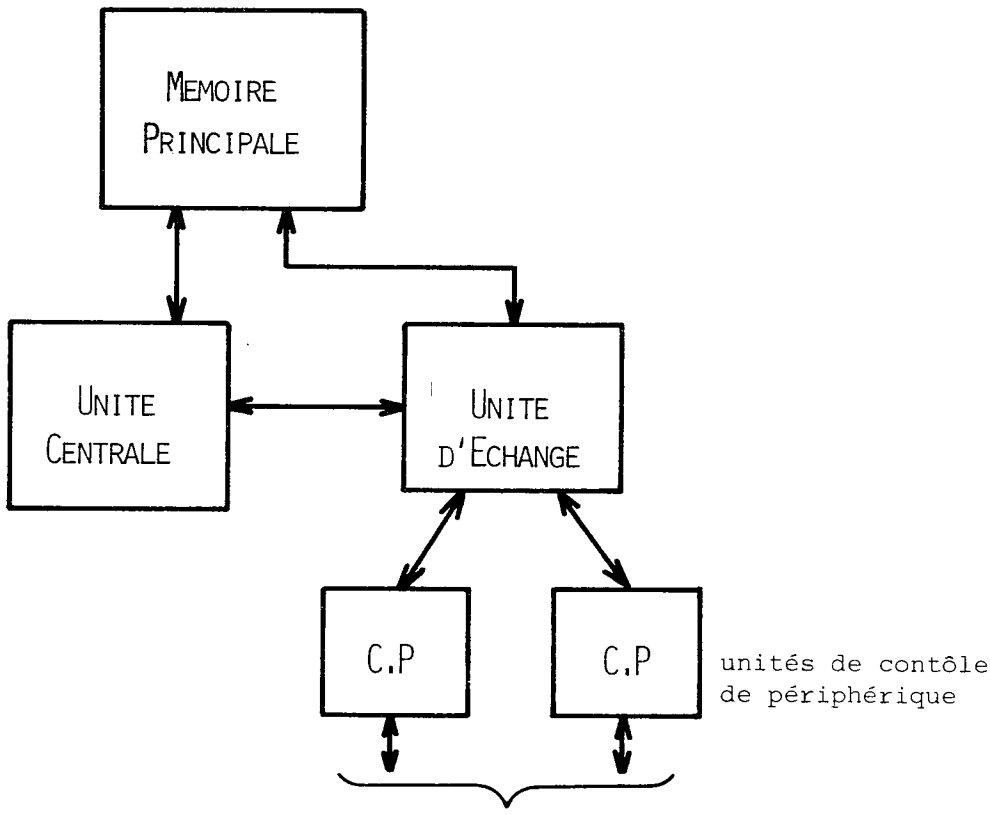


Figure II-7

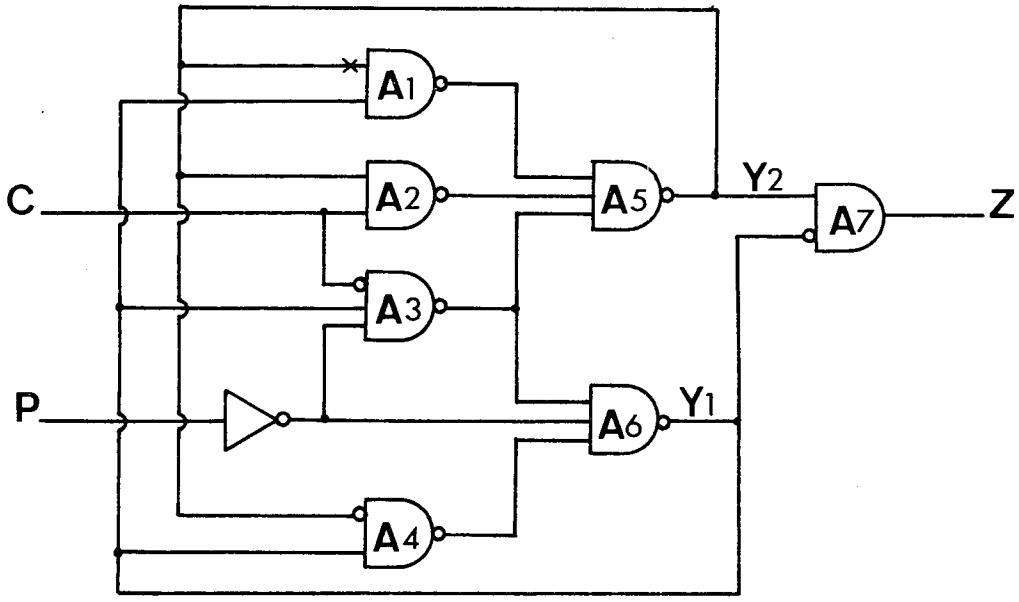


Figure II-8.a

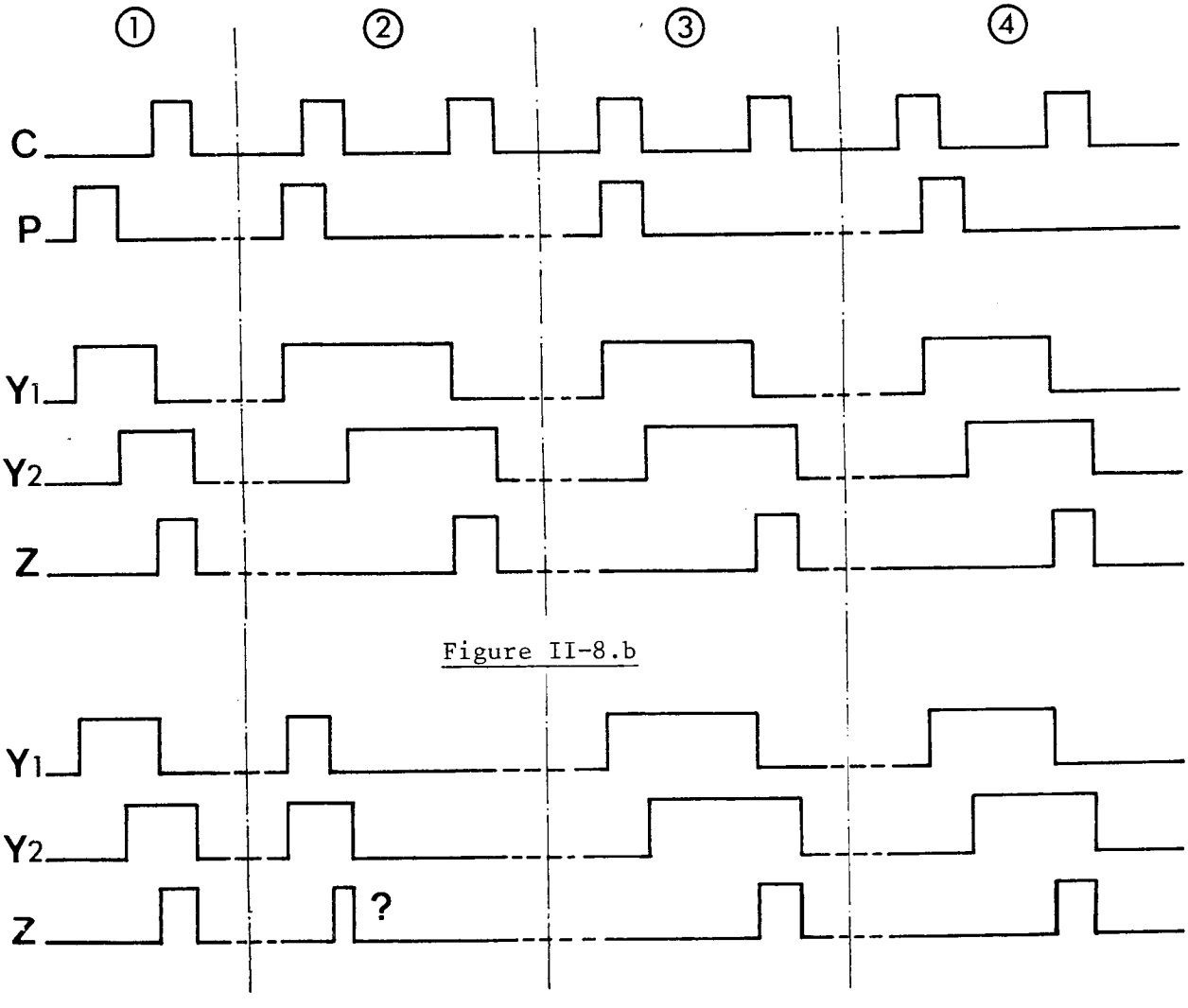


Figure II-8.b

Figure II-8.c

Les solutions ?

Le test hors ligne est, certes, la solution qui permet de résoudre de façon efficace le problème de la mise en oeuvre des tests dynamiques et des tests de simultanéité pour les circuits d'interface; encore que les tests dynamiques nécessitent des manipulations très précises que les outils dont on dispose généralement pour la maintenance sur site ne permettent pas de faire.

L'adoption d'une solution du type DSR (Chap.I, §III-3) facilite de façon considérable le test des circuits d'interface. On rappelle qu'une telle solution consiste à doter le système d'un processeur de test qui permet, par l'intermédiaire d'une liaison spéciale de maintenance, d'accéder à toutes les mémorisations essentielles des organes centraux du système sans avoir à utiliser les liaisons fonctionnelles standards. Le test des circuits d'interface peut alors être effectué de façon analogue à tous les autres circuits du système.

Pour un grand nombre des systèmes actuels, les tests des modules d'interface sont effectués selon des approches empiriques. Ces tests consistent à faire exécuter par le système (en bouclant un certain nombre de fois) des séquences de code judicieusement étudiées de manière à garantir, avec une forte probabilité, que cette exécution génère au moins une fois la séquence de test recherchée à l'interface du sous-système sous test. L'effet des pannes dans les circuits de synchronisation ou d'arbitrage se traduit soit par des erreurs dans l'exécution de ces séquences, soit par un écroulement sensible des performances du système. Les pannes dans les circuits de sélection des chemins de données créent des erreurs fréquentes dans les échanges d'information entre les sous-systèmes qui sont détectées soit par des vérifications logicielles effectuées par les séquences de test elles-mêmes, soit par analyse des enregistrements fournis par les circuits de recouvrement automatique d'erreur.

V - CONCLUSIONS

En faisant la distinction entre test statique et test dynamique, puis entre test unitaire et test de simultanéité, on a voulu poser clairement le problème du test des modules d'interface jusqu'ici mal défini. On a situé les difficultés qu'il pose tant pour la génération des tests localement pour ces modules, que pour leur mise en oeuvre depuis un interface de commande accessible par le système qui les englobe. Le tableau ci-dessous résume ces difficultés.

Certes, il s'agit là d'un problème qui se pose avec acuité pour des systèmes actuels, mais dont les dimensions pourront être considérablement réduites si l'on en prend meilleure conscience lors de la conception des futurs systèmes.

	Difficultés de génération du test au niveau du circuit	Composition du test pour sa mise en oeuvre au niveau système
Panne Unitaire Statique	Nulle	Faisable de façon déterministe
Panne Unitaire Dynamique	OUI Nécessité d'outils de simulation temporelle puissants pour la génération de chronogrammes de tests; ces outils manquent encore à l'heure actuelle	Impossible Nécessité de tests hors ligne sous outil
Panne de Simultanéité Statique	Nulle	Non faisable autrement que selon des approches empiriques
Panne de Simultanéité Dynamique	OUI Mêmes commentaires que pour les pannes unitaires dynamiques. De plus, la génération des tests pour ce type de panne n'est généralement pas automatisable	Impossible Nécessité de tests hors ligne sous outil

CHAPITRE III

TEST DES SOUS-SYSTEMES PERIPHERIQUES

I - PROBLEME DU TEST DES SOUS-SYSTEMES PERIPHERIQUES

La figure III-1 donne le schéma simplifié de la structure d'un sous-système périphérique. On peut distinguer trois parties :

- La logique d'adaptation du périphérique (D.A) assurant l'interface avec le coupleur;
- Les extensions électroniques de commande des mécanismes périphériques (D.O.E);
- L'unité électro-mécanique et électromagnétique (D.V).

Cet ensemble de constituants hétérogènes ne permet pas l'application d'une méthode générale de test ni un procédé algorithmique de génération de vecteurs semblable à ceux que l'on peut utiliser pour les ensembles logiques. Seuls des tests fonctionnels s'appliquant à l'ensemble du sous-système périphérique sont envisageables.

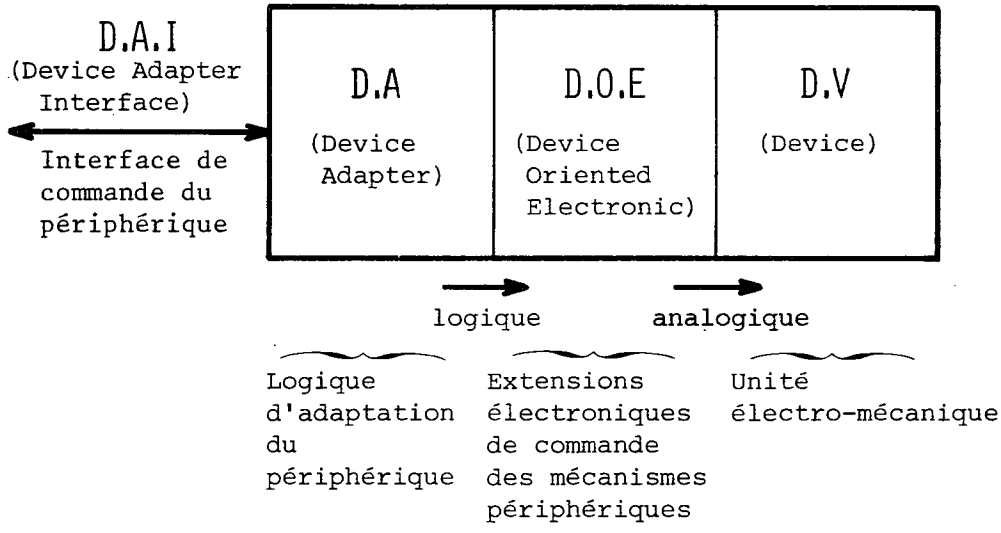


Figure III-1

Schéma de structure d'un sous-système périphérique

Une autre particularité des sous-systèmes périphériques est qu'ils ne sont pas fonctionnellement indépendants. En exploitation standard, ils sont activés par l'intermédiaire de processeurs de liaison avec l'unité centrale. Il faut que la vérification des fonctions du périphérique s'effectue dans des conditions d'activation similaires.

La mise au point en production bénéficie de la possibilité d'effectuer ces vérifications à l'aide de testeurs spécialisés. De la même manière que pour la vérification des sous-ensembles logiques, il est possible d'utiliser un système universel à condition d'étudier et de développer la logique d'adaptation propre à chaque type de périphérique.

La situation est tout autre en maintenance sur site. Dans un passé encore récent, la seule vérification possible consistait à faire exécuter par l'unité centrale du système, des programmes destinés à solliciter exhaustivement les fonctions réalisées par les périphériques; ceci permettant d'atteindre un certain niveau de diagnostic |C3||C4|.

Selon la nature et la finesse du test recherché, ce diagnostic est obtenu de diverses manières :

- par des contrôles externes effectués pendant le test en observant, par exemple, à l'oscilloscope l'allure des signaux en certains points internes prévus à ces fins. Généralement, ces points sont pris à l'interface de la partie logique avec l'électronique de commande ou à la sortie de l'électronique de commande vers les mécanismes du périphérique. Grâce à une documentation associée à la procédure d'observation, l'inspecteur de maintenance peut en déduire un diagnostic en comparant les courbes obtenues à celles attendues :

- Par le programme de test lui-même en analysant les bits d'état (Device Status) rendus par la circuiterie de contrôle interne du périphérique, ou à travers des vérifications effectuées sur les données transférées entre la mémoire principale et le support externe du périphérique (technique de lecture après écriture sur les unités à support magnétique par exemple).

- Dans certains cas, en constatant les résultats d'une opération effectuée par le périphérique comme par exemple le saut de page ou l'alignement et la netteté des frappes de caractères sur une imprimante. (Une frappe irrégulière peut résulter d'un déréglage, amenant une désynchronisation du mécanisme de déclenchement des marteaux avec la vitesse d'entraînement du tambour ou de la chaine de caractères).

Des efforts de plus en plus importants sont faits par les constructeurs de périphériques et par les responsables des services de maintenance afin de réduire les temps d'indisponibilité occasionés par le diagnostic et la réparation de ces dispositifs |C1||C2||C5||C6||C7| :

- En introduisant dans le sous-système périphérique un surplus de matériel de détection servant à automatiser le plus possible la tâche du diagnostic. Ces contrôles sont surtout importants dans les parties électroniques et électromécaniques où les fréquences de panne sont relativement plus importantes que dans la partie logique.

- En prévoyant le maximum de facilités logicielles (instructions ou procédures spécifiques au test, modes d'exploitation plus souples |C8|...) pour obtenir, des tests fonctionnels, le maximum d'efficacité et au coût minimum.

Ce n'est que très récemment que la généralisation des microdiagnostics, conséquence de l'apparition des unités de contrôle microprogrammées |C9||C10| |C11|, a permis de séparer véritablement deux niveaux de test :

- Les tests au niveau code orientés vers une vérification des spécifications fonctionnelles de l'interface logiciel-matériel;

- L'inspection fine et localisatrice réalisée par des microprogrammes commandés directement à partir des unités de contrôle des périphériques.

II - TECHNIQUE DU CONTROLE AUTOMATIQUE EN TEMPS REEL

La technique de l'aménagement des dispositifs de contrôle automatique dans les sous-systèmes périphériques ne s'est concrétisée de façon significative que dans des appareils relativement récents comme par exemple les unités bandes magnétiques 73341-73342 conçues par STC, les unités disque 3340 et 3350 conçues par IBM ou les imprimantes PR40 et PR70 d'Honeywell-Bull. Ces dispositifs fonctionnant en permanence, permettent d'effectuer trois types de contrôles :

α) Contrôles du type go-no-go

Ils ont pout but de contrôler l'intégrité de l'information et le séquencement correct des commandes élémentaires déclenchées par l'exécution d'une opération. Un ensemble de dispositifs permet de ramener ces contrôles à une détection de niveaux : courant, durée de temps, température,...

β) Contrôles marginaux

Ces contrôles interviennent en complément des contrôles go-no-go afin de permettre de détecter des conditions de fonctionnement marginal dûes à un début d'usure (vieillissement de composants électroniques ou électro-mécaniques) ou à une dégradation des paramètres d'environnement (alimentations, surface magnétique des supports externes...).

γ) Mesures de temps

Des dispositifs permettent de mesurer en permanence (à travers un ensemble de compteurs) le temps requis pour réaliser les opérations électro-mécaniques. Tandis que le contrôle go-no-go sert simplement à s'assurer qu'un certain signal d'entrée donne un signal de réponse bien défini, ces compteurs permettent par exemple d'enregistrer le temps qui sépare ces deux signaux. Ces compteurs peuvent être relevés directement par le logiciel et permettent de mettre en évidence des anomalies mécaniques ou une déviation à une tolérance établie.

L'exploitation de ces trois types de contrôle permet :

- de détecter les défauts générant des pannes permanentes ou intermittentes;
- de localiser le défaut dans une partie délimitée du matériel constituant le périphérique;
- de distinguer les effets liés à une défaillance dans l'électronique de commande de ceux dûs à des défaillances purement mécaniques ou électro-magnétiques;
- de déceler des conditions de fonctionnement marginal et donc de s'apercevoir de certains défauts dès leur naissance;
- de sécuriser, par les mesures de temps, l'ajustement des réglages électriques, électroniques ou mécaniques.

Ces dispositifs délivrent leurs sanctions par le positionnement d'information spécifique dans une mémoire propre au périphérique ("tampon détaillé d'état"). Cette mémoire peut être consultée par des moyens logiciels, et l'analyse de son contenu permet, d'une façon générale, d'en déduire un diagnostic précis.

L'expérience de Honeywell-Bull dans la réalisation des PR40 et PR70 (imprimantes) et CR500 (lecteur de cartes) pour la série 6400 a montré que la prise en compte des dispositifs de contrôle automatique, dès la conception, induit un surplus de coût qui ne dépasse guère 5 à 7% du prix de l'éléctronique et environ 10% du prix de la mécanique |C12|.

III - CLASSIFICATION GENERALE DES TESTS EXISTANTS POUR LES SOUS-SYSTEMES PERIPHERIQUES

On peut classer les tests existants selon quatre critères principaux :

- Le type d'action visé (détection, réglage, réparation);
- Le langage d'écriture;
- Le type de diagnostic fourni;
- Le mode d'exploitation.

Dans cette classification interviennent des critères secondaires tels que la fréquence d'emploi, et l'exhaustivité.

III - 1. CLASSIFICATION SELON LE TYPE D'ACTION VISEE

On distingue essentiellement 4 types de test :

1) Les tests préventifs de vérification fonctionnelle (Checkout Routines).

Ces tests sont orientés vers une vérification de conformité avec les spécifications fonctionnelles de l'interface logiciel-matériel. Ils sont destinés à être activés par l'opérateur, périodiquement à des fins de détection préventive, ou sur dépassement d'un seuil de qualité (fréquence importante de corrections automatiques sur un dérouleur de bande par exemple).

2) Les tests de recettes (Field Confidencing Tests)

Ces tests, plus complets, permettent de solliciter de façon exhaustive les diverses fonctions du périphérique et d'acquérir des renseignements plus sûrs sur son état de fonctionnement. Ces tests sont essentiellement utilisés par les inspecteurs de maintenance pour vérification après une intervention corrective ou périodiquement dans le cadre d'une maintenance programmée.

3) Les tests de diagnostic (Diagnostic Tests)

Ces tests ne sont activés que lorsqu'une panne a été détectée afin de la localiser et de procéder aux opérations de remplacement ou de réglage nécessaires. L'élaboration de ces tests est fondée sur une structuration et un ordonnancement rigoureux permettant d'obtenir, par recoupements et raffinements successifs, une localisation précise. Généralement, ces tests sont associés à un dictionnaire de pannes.

4) Les tests complémentaires de mesures (Mesurement Tests or Repair Aids)

On peut grouper dans cette classe tous les tests destinés à l'exécution et au contrôle des réglages électriques, électroniques ou mécaniques. Généralement, ces tests sont organisés en procédures paramétrables exécutées de façon itérative, et leur déroulement est commandé de façon interactive.

III - 2. LE LANGAGE D'ECRITURE DU TEST

Il varie du niveau langage machine (code ou micro-code) au niveau langages d'écriture de systèmes ou langages pour temps réel en passant par des traducteurs de tables de décision |C13|. On trouve aussi des langages libres, propres à chaque compagnie et pour chaque gamme; ces langages sont compilés |C14| ou interprétés au moment même du test |C15||C8|.

III - 3. LE TYPE DE DIAGNOSTIC

Un diagnostic peut être :

α) *Fonctionnel* : quand la présence d'une panne est signalée par l'erreur qu'elle engendre au niveau de l'interface logiciel-matériel.

β) *Référentiel* : lorsque la sanction délivrée est une clé d'entrée dans une procédure d'intervention (consultation d'un dictionnaire de pannes), lancement d'autres programmes, manipulations externes...).

γ) *Topologique ou Localisateur* : quand le programme de test spécifie la localisation de la panne dans un élément interchangeable.

δ) *Paramétrique* : quand le résultat du test est un ensemble de mesures permettant de caractériser le comportement de certains circuits ou mécanismes pris dans leur environnement.

III - 4. LE MODE D'EXPLOITATION

On note une évolution qui a bénéficiée de celle des logiciels d'exploitation (multiprogrammation, partage de ressources, reconfiguration du logiciel, contrôle interactif de l'exécution des tâches...) puis de celle de l'architecture des machines des nouvelles gammes(structures multiprocesseurs, unités de contrôle microprogrammées, processeurs de maintenance...). L'objectif étant

de rendre l'exploitation de ces tests plus souple, mais surtout de préserver au maximum les performances de l'exploitation continue du système supportant le périphérique |C3||C4||C15||C16||C14||C17||C8||C19||A14|.

On peut dresser la classification suivante :

1) Test autonome (Stand-Alone) : pour un programme de test exécuté seul sur machine nue.

2) Test sous-moniteur (off-Line Under Monitor) : pour un programme écrit de façon à être exécuté sous le contrôle d'un moniteur autonome, exploitation arrêtée.

3) Test sous-système altéré (On-Line) : pour un programme de test multi-programmé avec le système d'exploitation.

4) Test sous-utilisateur (Through-Line) : pour un test effectué à travers un processus utilisateur respectant les normes d'exploitation standard.

5) Test intégré (In-Line) : pour un test multiplexé avec les activités d'un processeur d'E/S capable d'un fonctionnement autonome pendant ses instants d'"oisiveté" (concentrateur de terminaux, unité de contrôle microprogrammée..).

6) Test hors ligne (Out-Line) : Lorsque le test est effectué par un processeur extérieur à la configuration du système supportant le périphérique (processeur de maintenance sur site, télémaintenance...).

IV - POLITIQUE DE MAINTENANCE DES SOUS-SYSTEMES PERIPHERIQUES ASSOCIEE A LA QUALITE DE FIABILITE ET DE FLEXIBILITE DU LOGICIEL D'EXPLOITATION

La qualité de "tolérance aux erreurs" d'un système (matériel et logiciel) se mesure par sa capacité d'entreprendre des actions spécifiques lorsqu'une erreur est détectée pendant l'exploitation. Les principaux objectifs sont :

- Effectuer des actions de reprise automatique lorsque celles-ci sont possibles (corrections, reconfiguration, repositionnement...);

- Faciliter les procédures de reprise sur incident non récupérable;

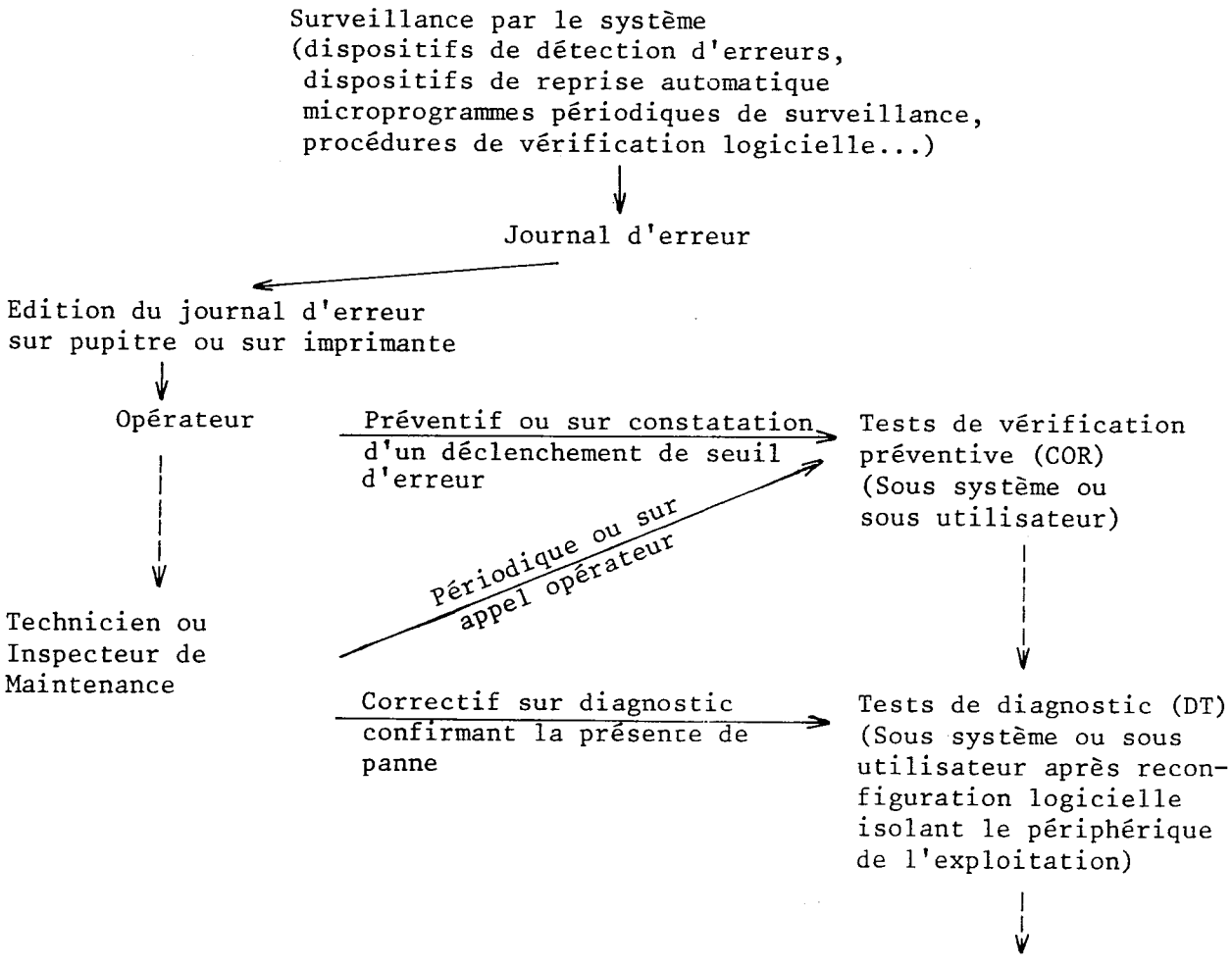
- Fournir des indications sur le comportement du système grâce à l'enregistrement, dans un journal d'erreur, des évènements qui présentent de l'intérêt du point de vue fiabilité et maintenance (logging).

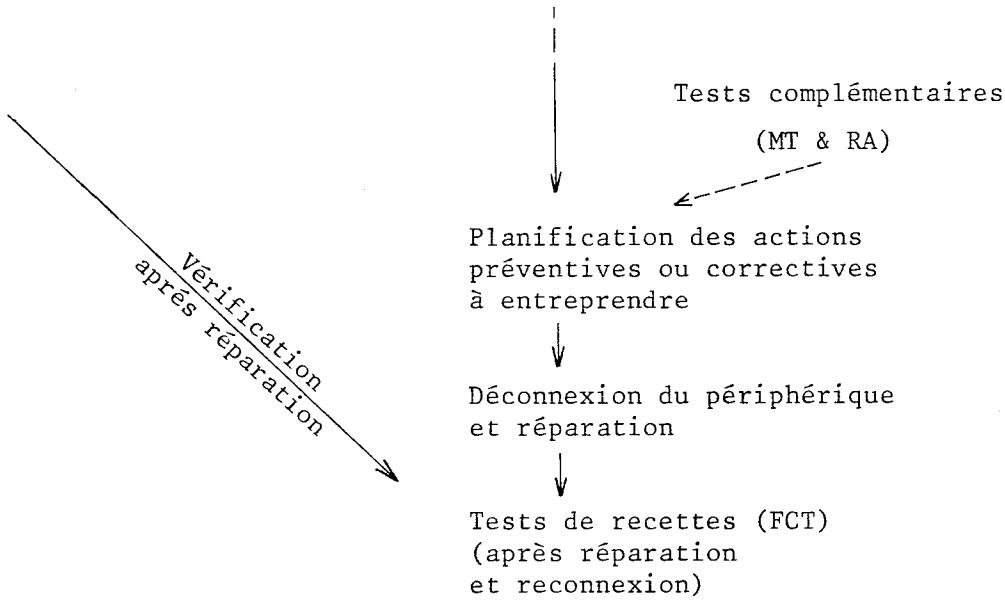
Pour les systèmes d'E/S, ces enregistrements peuvent être de deux types :

- Des enregistrements statistiques qui reflètent l'activité de chaque périphérique, par exemple : durée totale d'activité, nombre d'entrées/sorties effectuées, nombre d'octets transférés, nombre d'erreurs relevées...
- Des enregistrements de contextes d'erreurs : commande canal en cours au moment de la détection de l'erreur, identification du travail pour lequel les entrées/sorties sont effectuées, identification du volume en cas de périphérique à support magnétique, information d'état rendue par le canal et le périphérique, nombre de tentatives de reprise, erreur récupérée ou non...

La consultation du journal d'erreur pendant les heures d'exploitation permet à l'opérateur ou au technicien de maintenance de surveiller l'état de fonctionnement du système. A l'examen des enregistrements effectués, le dépassement d'un seuil d'erreur détermine l'exécution des programmes spécifiques de test afin de mieux connaître l'état du système et éventuellement de planifier les actions correctives à entreprendre en préservant au mieux la disponibilité du système.

On peut assister au scénario suivant :




Tests complémentaires
(MT & RA)
Planification des actions
préventives ou correctives
à entreprendre
Déconnexion du périphérique
et réparation
Tests de recettes (FCT)
(après réparation
et reconnexion)
Vérification
après réparation

CHAPITRE IV

LE PROGRAMME COSU

I - PRESENTATION DE COSU

COSU est un programme conversationnel de test sous-utilisateur de SIRIS 7/8 |C23| des périphériques supportés par les systèmes CII 10070 et IRIS 80 |C20||C21||C22|.

Avant l'élaboration de cet outil, la maintenance des périphériques se faisait sous moniteurs de test en off-line |C15|. Les seuls tests sous-utilisateurs disponibles avaient été écrits pour traiter des problèmes particuliers, par des spécialistes de maintenance ayant une connaissance approfondie des deux aspects logiciel et matériel de ces systèmes. Le projet COSU a eu pour but la généralisation du concept du test sous-utilisateur et la définition de l'environnement nécessaire à une exploitation aisée par les inspecteurs de maintenance.

Le programme COSU est un produit expérimental qui permet de façon interactive, l'écriture en termes de primitives de haut niveau, la mise au point, le cataloguage et l'exécution de programmes de test des périphériques. Certains résultats ont contribué à l'intégration officielle des tests sous-utilisateurs dans la récente version C10 de SIRIS 7/8.

II - CONCEPTS DE BASE DU TEST FONCTIONNEL DES PERIPHERIQUES

On appellera *Protest* une séquence de code (ou procédure) rassemblant un certain nombre d'instructions, de programmes canaux et de données judicieusement choisis et agencés de façon à ce que son exécution permet d'effectuer le test d'une fonction d'un périphérique.

Un programme de test pour un périphérique est un ensemble ordonné de protests.

L'activation d'un périphérique, en connexion standard, s'effectue à travers une unité de liaison (UL) elle-même commandée par une unité d'échange (UE ou canal) Figure IV-1. Un programme de diagnostic du périphérique doit s'assurer du bon fonctionnement de ces unités pendant le déroulement du test. L'élaboration des protests, leur séquencement et la *dynamique* de leur exécution doit en tenir compte (à moins que le test des UL et canaux puisse se faire séparément, autrement que par des protests).

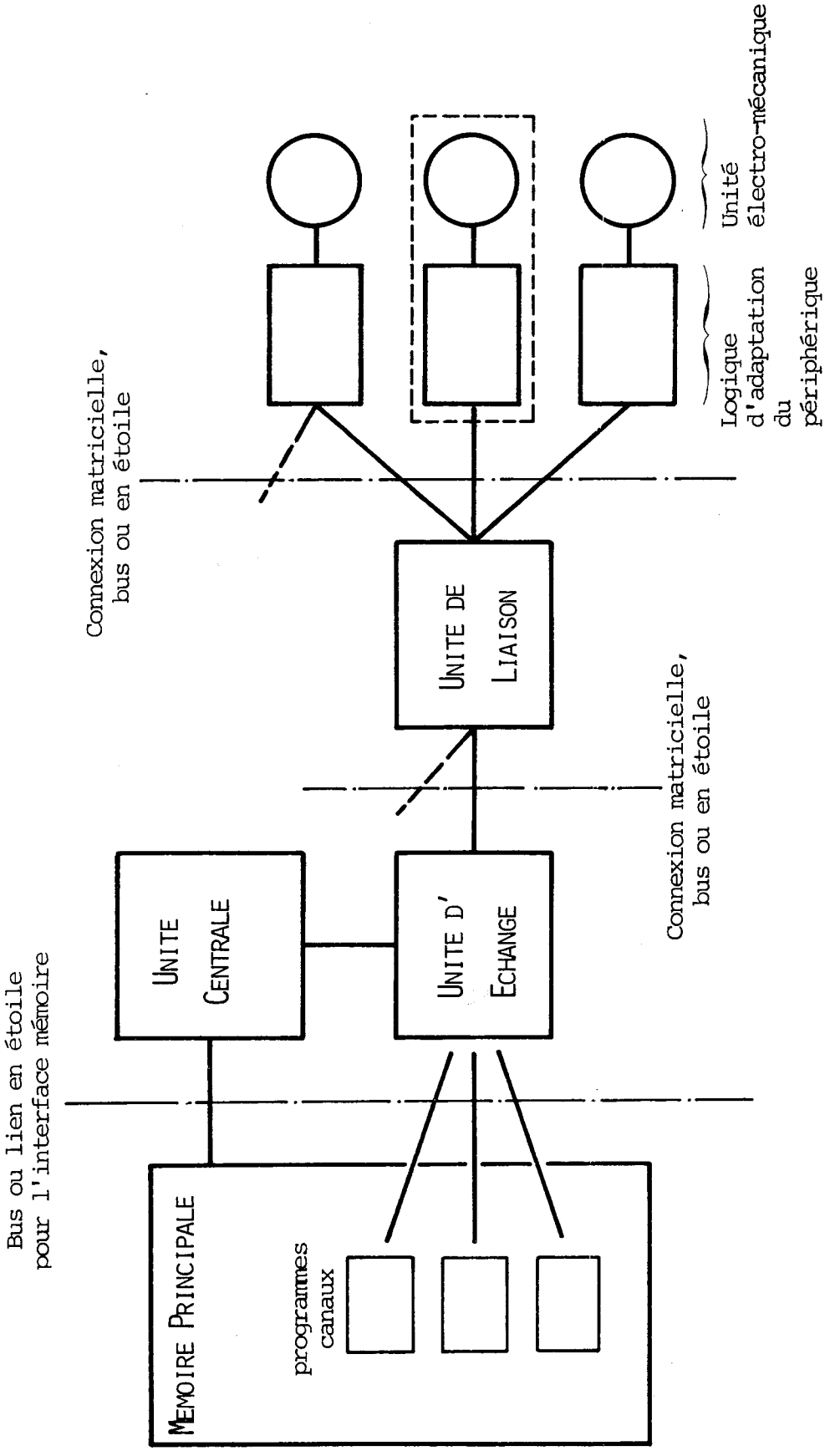


Figure IV-1

Schéma de connexion des périphériques

Pour l'intégration des protests avec l'exploitation, les contraintes suivantes apparaissent :

1) Les entrées/sorties sont exécutées en mode privilégié.

2) Les transferts s'effectuent obligatoirement à des adresses mémoire réelles et contigües.

3) La dynamique de test des périphériques peut dépendre du temps de réponse des unités de liaison, canal et mémoire concernées par cette exécution, donc du taux de charge de ces unités. Dans ce cas, le test nécessite, non seulement la réservation de ces unités, mais également la maitrise du temps d'exécution des opérations d'entrées/sorties.

Résoudre les problèmes de la mise en oeuvre des tests des périphériques en parallèle à l'exploitation revient donc à fournir les moyens qui permettent de satisfaire à ces trois contraintes.

Dans la solution du test sous-système, on plie le logiciel d'exploitation aux exigences d'un moniteur de test autonome. Les techniques développées pour la multiprogrammation, le partage des ressources physiques, la reconfiguration dynamique du logiciel, l'exploitation temps réel... offrent une multitude de solutions possibles.

Dans la solution du test sous-utilisateur, on crée un interface par lequel le moniteur de test, tout en se comportant comme une tâche utilisateur mais privilégiée, peut forcer la création de l'environnement nécessaire pour le déroulement du test.

Le choix de l'une ou l'autre des deux solutions dépend des caractéristiques du système d'exploitation et du sous-système périphérique concernés. On peut cependant souligner, dans le cas des tests sous-utilisateur, l'intérêt présenté par les possibilités d'analyse du journal d'erreur et des comptes-rendus d'exécution, ainsi que de composition de séquences de test adaptées de façon interactive et suivant la même syntaxe que celle du dialogue normal de commande du système d'exploitation. L'introduction de ces possibilités est en effet plus avantageuse dans le cas des tests sous-utilisateur que dans le cas des tests sous-système puisque, dans le premier cas, la plupart des outils nécessaires (système de gestion des transmissions pour la connexion d'une console de dialogue, système de gestion de fichiers pour le cataloguage des protests...) sont déjà disponibles, alors qu'il faut complètement les réécrire pour le cas du test sous-système (le moniteur étant autonome).

En plus du travail que cela nécessite, l'accroissement inévitable du volume de code est de nature à léser davantage les performances de l'exploitation.

Bien sûr, toutes ces considérations ne peuvent être retenues que pour des systèmes ne disposant pas des possibilités du test "in-line" ou "out-line".

III - ELEMENTS DE L'INTERFACE A DEVELOPPER POUR LE TEST SOUS-UTILISATEUR

Il importe de faire la distinction entre les tests dont la dynamique d'exécution a relativement peu d'importance (tests de vérification statique) et ceux pour lesquels la prise en compte du facteur temps est une nécessité de premier ordre (tests visant à s'assurer du bon fonctionnement des UL et canaux ainsi que du fonctionnement dynamique du périphérique).

Les éléments de l'interface nécessaire pour satisfaire les contraintes d'exécution pour les tests du premier type sont les suivants :

1) La possibilité pour un programme utilisateur privilégié de se réserver l'utilisation exclusive d'une unité périphérique dont il fournit l'adresse.

2) Une procédure système qui permet à cet utilisateur de commander directement, au niveau de l'interface canal, le périphérique qu'il s'est réservé et de récupérer toutes les informations de réponse rendues par ce périphérique.

3) Enfin, la possibilité pour ce programme de s'octroyer un espace mémoire réelle qui lui sera réservé et figé tout le long de son exécution et pour lequel il est autorisé à tous les accès.

Ces possibilités sont plus ou moins offertes par la plupart des systèmes d'exploitation qui prévoient la prise en charge d'applications type temps réel :

1) Généralement, la réservation des unités périphériques est accordée de façon statique (une fois pour toutes, au moment du chargement pour exécution du programme). C'est notamment le cas sous SIRIS 7/8 |C23|. Ceci constitue une entrave à l'optique du test préventif qui englobe plusieurs périphériques, puisqu'on est obligé d'immobiliser inutilement tous ces périphériques dès le moment du chargement du moniteur de test, sinon, réeffectuer ce chargement pour chaque sous-groupe, voire pour chaque périphérique. Quand elle

est possible, la réservation dynamique (éventuellement arbitrée par le pupitreur) offre une meilleure souplesse pour l'exploitation des tests. La réservation dynamique est nécessaire si l'on veut trouver une solution pour le test des périphériques système (console opérateur, disque système...) chose qui n'est pas possible avec la réservation statique.

2) Une procédure système type EXCP (EXecute Channel Program) est généralement prévue pour permettre à des utilisateurs privilégiés dits " de la classe d'exploitation temps réel" de se faire connecter des périphériques non-standards (convertisseurs, machines-outils...) via les canaux du système. Sous SIRIS 7/8, la procédure M:EXCP s'étend à tous les périphériques, et son utilisation, conjuguée avec celle de M:ATTACH |C23| pour la récupération directe des interruptions, permet de programmer les entrées/sorties au niveau canal et de contrôler l'exécution de ces opérations tout en étant sous mode utilisateur.

On peut regretter, cenpendant, certaines lacunes qui sont limitatives de l'efficacité du test. Les remarques que l'on va faire de façon spécifique pour la procédure M:EXCP sous SIRIS 7/8 (v. B08A/C01A) peuvent être généralisées à toutes les procédures type EXCP dans les autres systèmes.

a) A partir de l'instant où l'exécution de la procédure M:EXCP est demandée jusqu'à l'arrivée de l'interruption associée à la fin de l'entrée/sortie, la tâche de test n'est pas active, et il n'est pas possible de programmer des interruptions intermédiaires. Ceci interdit le prélèvement d'informations d'état du périphérique pendant l'exécution de l'entrée/sortie.

b) Ce type de procédure exploite l'adressage d'un périphérique par son étiquette logique qui ne traduit pas la voie d'accès physique qui sera effectivement utilisée pour l'opération d'entrée/sortie. Cet inconvénient est particulièrement sensible dans le cas d'une connexion matricielle. Aussi, elle ne traduit pas de façon précise le type de périphérique et ne permet pas de contrôler la validité des ordres d'E/S composés de façon interactive.

c) Enfin, cette procédure ne prévoit pas le prélèvement des informations d'état détaillées que certains périphériques fournissent pour le diagnostic et donc ne permettent pas d'en faire profiter l'analyse effectuée dans le programme de test.

3) Dans les systèmes travaillant en adressage virtuel (cas de SIRIS 7/8), il n'est pratiquement pas possible d'obtenir un espace mémoire réelle qui soit contigü; la contiguité n'est garantie que dans les limites de la taille des pages mémoire (2 K octets). Il en résulte que l'on ne peut pas commander des transferts d'une longueur supérieure à une page sans faire de chainage de données. Les programmes canaux, eux-aussi, doivent être programmés dans une même page mémoire ou alors chaînés par des commandes de saut-canal. Toutefois, cette contrainte a une influence négligeable sur le test du périphérique étant donné que l'exécution des chainages n'incombe que le canal et l'unité de liaison.

Pour les tests du deuxième type qui requièrent une maîtrise absolue de la dynamique d'exécution des programmes canaux, l'interface avec le système d'exploitation doit permettre de plus :

4) D'étendre les possibilités de réservation aux unités de liaison et canaux.

5) De donner la possibilité au programme utilisateur qui effectue le test d'intervenir à tout moment dans le déroulement des opérations d'entrées/sorties; en particulier, le droit de programmer des interruptions intermédiaires et de commander l'arrêt d'une opération d'E/S en cours.

6) De mettre à la disposition de ce programme un compteur temps réel de fréquence suffisante pour permettre des mesures significatives des temps d'exécution des opérations d'E/S ou le cadencement précis du lancement de ces opérations. La possibilité d'effectuer des mesures implique inévitablement la complicité du système d'exploitation pour la remise à jour ou le prélèvement de ce compteur sur déclenchement d'évènements déterminés lors du déroulement des opérations d'E/S.

Ces extensions que l'on vient d'énumérer sont envisageables pour SIRIS 7/8 mais n'existent pas dans les versions opérationnelles à ce jour.

Statistiquement, si l'on considère le nombre de pannes possibles pour un périphérique, on constate que 15% à 35% de ces pannes ne sont pas prises en compte par les programmes de test; soit parce que leur mise en évidence est difficile ou nécessite un temps d'exécution trop long (8 à 15% des cas) et l'on préfère alors renvoyer le technicien de maintenance à des procédures d'intervention utilisant des moyens externes; soit parce qu'elles sont rares (5 à 12% des cas) et l'on fait exprès de ne pas en tenir compte afin de ne

pas trop alourdir les tests (on peut les programmer dans des tests séparés de derniers recours); soit simplement parce qu'on les a omises ou négligé leurs effets (2 à 17% des cas), d'où l'intérêt de pouvoir composer des séquences de tests adaptées de façon interactive,au moment du test.

Pour les pannes testées, on peut considérer que 50% à 80% (selon le type de périphérique) sont détectables par des tests de vérification statique. C'est pourquoi, en dépit des limitations que nous avons fait constater plus haut pour les moyens de tests offerts par SIRIS 7/8, il a été décidé de tenter l'expérience de la réalisation de COSU.

IV - DESCRIPTION SUCCINCTE DE COSU

Il n'est pas possible de reprendre ici tous les détails de la réalisation de COSU; on se reportera à la documentation référencée en |C20||C21||C22||C19|.

On se limitera à présenter l'architecture globale de ce programme et à décrire quelques unes des particularités de ses modules.

Le but de COSU est de permettre, de façon interactive sous SIRIS 7/8, la création de protests avec tout ce que cela comporte comme procédure d'écriture, de correction, d'assemblage, de cataloguage,..., puis l'exécution toujours interactive de ces protests. Il a été décidé d'organiser son architecture en deux sections principales, chaque section regroupant un certain nombre de modules (Figure IV-2).

La section d'Edition EDI

Elle concerne tout ce qui se rapporte aux fonctions de dialogue et de création de protests; son exécution ne requiert aucun des services privilégiés de SIRIS 7/8. Elle comprend :

- Le module DECODEUR pour la gestion des entrées/sorties sur la console de dialogue et le décodage des commandes d'entrée. Ce module bénéficie des procédures offertes par le système de gestion des transmissions (SGT) sous SIRIS 7/8 |C24|.

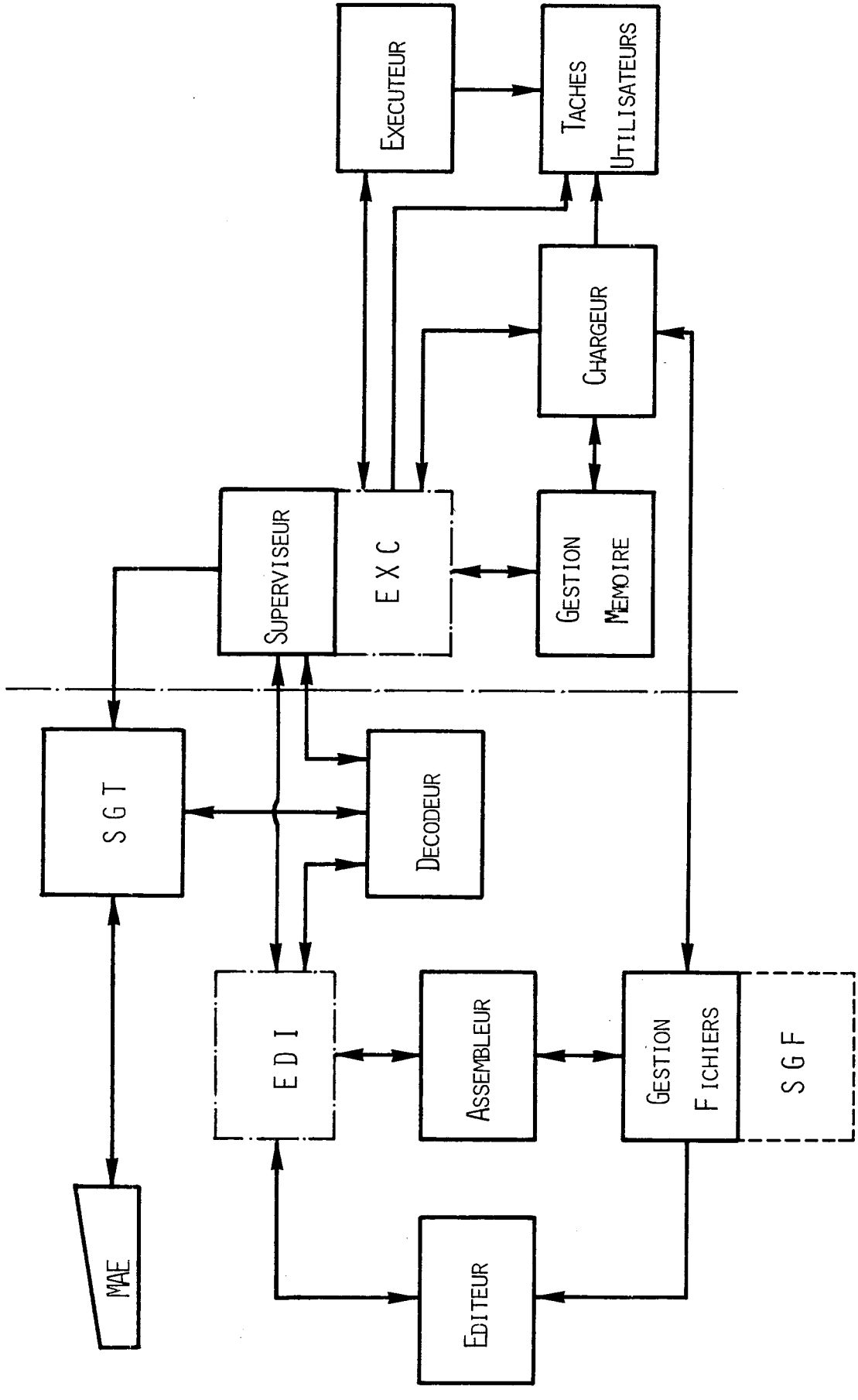


Figure IV-2

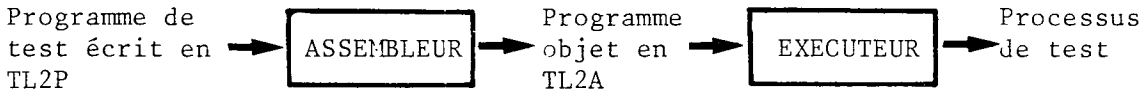
Structure du programme COSU

- Le module EDITEUR pour l'entrée, la correction et l'édition des programmes de test. Chaque programme est constitué comme un fichier occupant une partition dans une bibliothèque de tests. Le module GESTION-FICHIERS gère cette bibliothèque en utilisant les facilités offertes par le système de gestion des fichiers (SGF) sous SIRIS 7/8 |C25|.

- Le module ASSEMBLEUR pour l'assemblage des programmes de test entrés en langage symbolique et la création des tables nécessaires à leur exécution. Un même programme peut être rangé en bibliothèque sous ses deux versions externe et assemblée.

Afin de rendre l'écriture des protests aisée et rapide, il était indispensable de concevoir un langage externe (TL2P)*1 regroupant des primitives de haut niveau. L'analyse des besoins a montré que l'on peut se suffire d'une vingtaine de primitives |C20|. Une procédure faisant partie du module EDITEUR permet, à la demande de l'utilisateur, de contrôler systématiquement la construction correcte des programmes canaux selon le type de périphérique concerné. Des entrées erronées telles qu'un ordre incompatible avec le type du périphérique ou un indicateur illégal... sont détectées et signalées au moment de leur entrée; d'autres erreurs moins importantes telles qu'un compte d'octets trop grand ou un chainage de commande non satisfait.. sont acceptées par l'EDITEUR mais seront signalées lors de l'assemblage par la sortie de messages d'avertissement appropriés. Ces contrôles contribuent ainsi à garantir le bien fondé d'un diagnostic en éliminant au maximum les erreurs "non volantaires" dans la programmation des protests.

La solution de l'assemblage direct en code machine fut envisagée puis écartée; elle pose le problème de la protection de COSU contre les éventuelles erreurs d'exécution des protests. Il a été décidé d'assembler ce langage dans un code intermédiaire (TL2A) possédant les caractéristiques d'un code machine et interprété à l'exécution par un module EXECUTEUR.



Le module EXECUTEUR fait partie de la deuxième section EXC.

---

*1 - Through Line Test Language; la lettre adjointe est significative de l'usage de cette partie du langage : P pour programmation, A pour assemblage et C pour commande.

La section d'Exécution EXC

C'est la partie du programme COSU, qui a pour fonction de créer, sous SIRIS 7/8, l'environnement nécessaire à l'exécution des tests, puis de lancer et contrôler l'exécution de ces tests.

- Lorsque l'utilisateur désire faire exécuter un programme de test, il en avertit COSU par une commande du sous langage TL2C. Le module CHARGEUR vérifie que ce programme figure en bibliothèque sous forme assemblée puis, en se servant des tables préparées par l'ASSEMBLEUR à l'entête du code objet, il effectue les fonctions de chargement nécessaires et établit l'interface d'exécution avec le module EXECUTEUR. Le rôle du CHARGEUR est compliqué par les problèmes de gestion de mémoire résultant de la non-contiguité de l'espace réel alloué par SIRIS 7/8 (voir remarque § III.3).

- Un programme de test dont l'exécution a été initialisée devient une *tâche de test.* La conception de COSU autorise, grâce à un module SUPERVISEUR, l'exécution parallèle de plusieurs tâches de test à condition qu'elles concernent des périphériques différents. Cela permet, en des moments déchargés de l'exploitation, d'optimiser les temps d'exécution des tests préventifs.

- Enfin, le module EXECUTEUR est chargé de l'interprétation proprement dite des instructions du langage TL2A. Le langage TL2P prévoit des primitives qui permettent de programmer *des points de reprise* et des *points d'arrêt* systématiques ou conditionnels. Grâce à ces repères, l'utilisateur peut intervenir lors du déroulement de la tâche de test pour demander la réitération de l'exécution à partir d'un point de reprise donné, la poursuite de l'exécution de là où elle a été arrêtée, ou la suspension définitive de la tâche de test.

V - CONCLUSIONS

La réalisation de COSU, malgré l'insuffisance des résultats attendus, fut un pas qui valait la peine d'être franchi à un moment où les coûts de développement et d'exploitation des tests de périphériques commençaient à prendre des proportions inacceptables dans les prix des systèmes. Elle a eu le mérite de décortiquer les problèmes qui s'opposent à l'intégration de ces tests sous utilisateur ou sous système et de proposer des suggestions concrètes et chiffrées qui ont aujourd'hui apporté leur fruit.

Comme on l'a souligné dans le chapitre précédent, un effort important tend à faire disparaître les contraintes de temps qui sont apparues comme rédibitoires dans la vérification des sous-systèmes périphériques des générations précédentes. Ceci en mettant en place dans les différentes unités de contrôle des périphériques, des moyens de vérification temps réel et de stockage de paramètres permettant de fournir à l'unité centrale qui dirige les tests des informations déjà élaborées et pouvant être analysées en différé. De plus, une tendance générale conduit à doter ces sous-systèmes périphériques de dispositifs de rebouclage avec des enregistrements simulés marginaux permettant la vérification des détections de seuil et des dispositifs de contrôle et de correction. Ces tendances font qu'une part importante des tests des périphériques peut être conduite dans un temps minimum en partage avec une exploitation standard, et justifie la réalisation de ces tests sous système d'exploitation. Toutefois, certains points de fonctionnement ne peuvent être analysés simplement par des programmes non paramétrés. C'est le cas, en particulier, de la vérification des média disque ou bande magnétique. C'est également le cas de certaines fonctions électroniques ou électromagnétiques dont un mal fonctionnement nécessite un réglage.

Selon ces tendances, on distingue trois types de tests :

- Le test des sous-ensembles logiques qui sont réalisés par des diagnostics, généralement intégrés à l'unité. L'élaboration de ces tests est identique à celle des tests pour les ensembles logiques synchrones.

- Le test des sous-ensembles analogiques ou à fonctionnement logique asynchrone (chaine de lecture-écriture d'un dérouleur de bande ou d'une unité disque par exemple) qui sont effectués par l'intermédiaire des dispositifs de rebouclage et dont l'exécution est automatique mais nécessite un temps d'immobilisation important.

- Le test des média (disque, bande) ou des dispositifs électroniques ou pneumatiques pour lequel le paramétrage est nécessaire; en particulier, afin de corriger des réglages.

Le premier type de test de durée courte peut ou non être exécuté sous-système d'exploitation. Les deux derniers types de tests doivent être faits en partage avec l'exploitation si on veut limiter les temps d'immobilisation

du système. En outre, des possibilités de paramétrage et de choix des séquences de test doivent être prévues sous-système d'exploitation pour permettre l'exécution des derniers types de test.

Le programme COSU suggère, dans ce sens, des solutions intéressantes.

Ont également contribué à la réalisation de ce programme, Messieurs :
P. DECITRE, du Centre Scientifique CII à Grenoble,
A. JORRY, actuellement à l'Institut für Informatik der T.U.M, München - RFA,
P. ROUX, du Service D.A.V-CII à Grenoble.

CHAPITRE V

TESTS DE VERIFICATION FONCTIONNELLE EN INTEGRATION DES SYSTEMES. TESTS D'ACCEPTATION

I - INTRODUCTION

On a jusqu'à présent abordé la vérification des dossiers matériels qui consiste à tester l'absence de défauts dans les sous-ensembles matériels constitutifs d'un système. Le terme sous-ensemble est utilisé ici dans son sens large; ce peut être aussi bien un composant simple, une carte équipée ou un sous-système. On assiste actuellement à une prise de conscience du problème de la maintenabilité des systèmes dès leur conception; en particulier, on met en place des stratégies qui permettent de tester localement les divers sous-ensembles sans devoir composer les tests au niveau système intégré.

La vérification des dossiers fonctionnels consiste à contrôler que ces sous-ensembles s'intègrent correctement pour réaliser les fonctions globales définies pour le système. Ces tests sont effectués essentiellement en fin de mise au point de prototype, afin de vérifier que le système nouvellement conçu et implémenté réalise les objectifs de fonctionnement et de performances attendus. Ils sont aussi effectués en phase d'intégration sur plateformes de fabrication de série et au moment de l'installation sur site, à des fins d'essai préventif après assemblage et réglages. Toutes les configurations et options possibles du système ne peuvent être testées au stade du prototype; d'autre part, les configurations requises par les clients sont rarement identiques, ce qui justifie la nécessité d'accorder aux tests effectués sur plateformes, la même importance qu'à ceux effectués en mise au point du prototype. Les tests effectués au moment de l'installation sur site peuvent être relativement plus sommaires.

Les tests d'acceptation dont on a introduit les objectifs dans le chapitre I (§ II-2) s'inscrivent parmi les tests effectués dans le cadre de la vérification des dossiers fonctionnels. On rappelle qu'il s'agit d'un ensemble de programmes qui permettent d'automatiser certaines vérifications et faciliter certains réglages assurant la bonne intégration des diverses unités fonctionnelles du système. Utilisés dans le contexte spécifique des tests d'intégration sur plateformes, ils permettent, par bouclage intensif, d'effectuer la phase de déverminage.

Alors que la littérature universitaire et industrielle est abondante pour ce qui concerne la vérification des dossiers matériels, la vérification des dossiers fonctionnels est restée dans l'ombre. On se propose dans ce chapitre de formaliser ce problème et de suggérer des solutions concernant :

- La recherche d'un modèle adéquat de description des fonctions du système facilitant la compréhension du lien existant entre une description algorithmique de ces fonctions et le matériel qui les implémente;

- L'élaboration, à partir de ce modèle, de méthodes de test les plus systématiques et les plus réalistes possibles;

- L'étude de ces méthodes.

Dans le chapitre suivant, on s'intéressera particulièrement aux tests d'acceptation et à l'outil logiciel dont on doit disposer afin de faciliter la mise en oeuvre de ces tests.

II - MODELE DE DESCRIPTION DES FONCTIONS-SYSTEME

II - 1. LES FONCTIONS-SYSTEME

Les fonctions globales réalisées par un système peuvent être classées en trois catégories :

- Les fonctions programmables au niveau de l'interface logiciel-matériel à l'aide des instructions du langage machine. On notera l'ensemble de ces fonctions par F_i.

$$F_i = \{f_{i_1}, f_{i_2}, \ldots f_{i_p}\};$$

- Les fonctions commandables au niveau de l'interface homme-machine moyennant des dispositifs de commande tels que le panneau d'exploitation, les panneaux de reconfiguration, etc... On notera l'ensemble de ces fonctions par F_c

$$F_c = \{f_{c_1}, f_{c_2}, \ldots f_{c_q}\};$$

- Enfin, les fonctions spéciales de test qui sont activées et contrôlées à partir d'un processeur de maintenance (dans le cas où le système est muni d'un tel dispositif). On notera l'ensemble de ces fonctions par F_m.

$$F_m = \{f_{m_1}, f_{m_2}, \ldots f_{m_r}\}.$$

On notera $F^o = F_i \cup F_c \cup F_m$. F^o constitue l'ensemble des fonctions-système.

Remarque : Dans la suite, on excluera de F^o toute fonction liée uniquement à l'implémentation matérielle et n'ayant aucune interprétation algorithmique pour l'utilisateur. (Exemple : fonctions spéciales de lecture ou d'écriture de points de test spéciaux, utilisant des voies d'accès autres que les voies fonctionnelles).

Pour réaliser ces fonctions globales, le concepteur a dû être amené à créer un nouvel ensemble F^1 de sous-fonctions telles que chaque fonction de F^o puisse être réalisée à partir d'algorithmes utilisant ces sous-fonctions. Il les a ensuite réparties en groupes et implémenté chaque groupe dans un même module matériel.

Considérons, à titre d'exemple, le cas d'une fonction-instruction. L'exécution de cette fonction requiert d'une façon générale la séquence d'opérations suivante :

1) Acquisition du contexte de l'instruction;
2) Transcodage de l'instruction et calcul des adresses des opérandes;
3) Recherche des opérandes;
4) Exécution;
5) Notification des résultats obtenus.

Ces opérations sont réparties entre des unités mémoire (1 et 3), des processeurs de préparation et d'anticipation au traitement des instructions (2 et 5), des opérateurs spécialisés (4) etc..

Les fonctions de chaque module ont du, elles-mêmes, être réalisées à partir de sous-fonctions d'un niveau plus fin, réparties entre divers sous-modules. Au niveau le plus bas, le concepteur a associé des composants (portes, boitiers, microprocesseurs,...).

Cette hiérarchie F^1, F^2, F^n.. de sous-fonctions est décrite dans les dossiers fonctionnels et illustrée à l'aide de synoptiques et d'organigrammes suffisamment détaillés pour permettre de comprendre les algorithmes utilisés dans l'implémentation des fonctions-système.

II - 2. CRITERES DE CHOIX DU MODELE

La vérification des dossiers fonctionnels vise deux objectifs principaux :

1) Le premier objectif est de révéler des situations d'exécution des fonctions-système qui ont été incorrectement résolues ou non prévues. Seule l'approche du test exhaustif permet de réaliser un tel objectif. Cependant, l'extrême difficulté rencontrée dans l'élaboration d'une stratégie de test exhaustif pour les grands systèmes, les temps exorbitants que demanderait l'exploration en détail de toutes les possibilités des fonctions du système rendent cette approche pratiquement inenvisageable.

Une approche plus réaliste consiste à expérimenter chaque fonction du système pour un nombre limité de contextes d'exécution jugés représentatifs.

Le choix du modèle doit faciliter et justifier la sélection de ces contextes.

2) Comme on l'a déjà mentionné en chap.0, les dossiers fonctionnels comportent deux types de documents :

- . Les manuels utilisateurs qui décrivent le système vu de l'extérieur comme une boite noire qui réalise des fonctions globales;
- . Les spécifications de réalisation qui décrivent les détails des algorithmes qui ont été adoptés dans l'implémentation de ces fonctions.

Les utilisateurs se réfèrent généralement uniquement au premier type de documents. Aussi, un deuxième objectif de la vérification des dossiers fonctionnels est de s'assurer que ces manuels donnent une description aussi exacte et complète que possible des fonctions telles qu'elles sont réalisées par le système. Une description confuse ou incomplète réserve parfois des surprises désagréables pour l'utilisateur.

Le modèle choisi doit permettre le contrôle de la cohérence entre les deux types de dossiers.

L'approche proposée consiste à utiliser une description intermédiaire sous forme d'une "table de correspondance causes-effets".

- Les causes correspondent aux valeurs de conditions logiques sur les différents paramètres qui déterminent le déroulement de l'exécution des fonctions-système . Ces paramètres peuvent être classés en trois catégories :

C1 : Les codes-opération des fonctions avec les paramètres d'entrée correspondants : pour chaque fonction, ces paramètres spécifient les options d'exécution et les données sur lesquelles elle porte. (Pour une fonction-instruction, ce sera par exemple une indication du type d'adressage direct ou indirect, avec ou sans indexation, les données concernées par l'exécution de l'instruction, etc...).

C2 : Les paramètres d'environnement qui déterminent la configuration opérationnelle du système et son mode de fonctionnement : ce sont par exemple des paramètres que l'on peut introduire en positionnant des clés sur les panneaux de commande (reconfiguration, exécution pas-à-pas, masques, arrêt sur adresse...).

C3 : L'état du système représenté par le contenu de certains éléments de mémorisation (registres d'état, registres généraux, code-conditions, mots mémoire...).

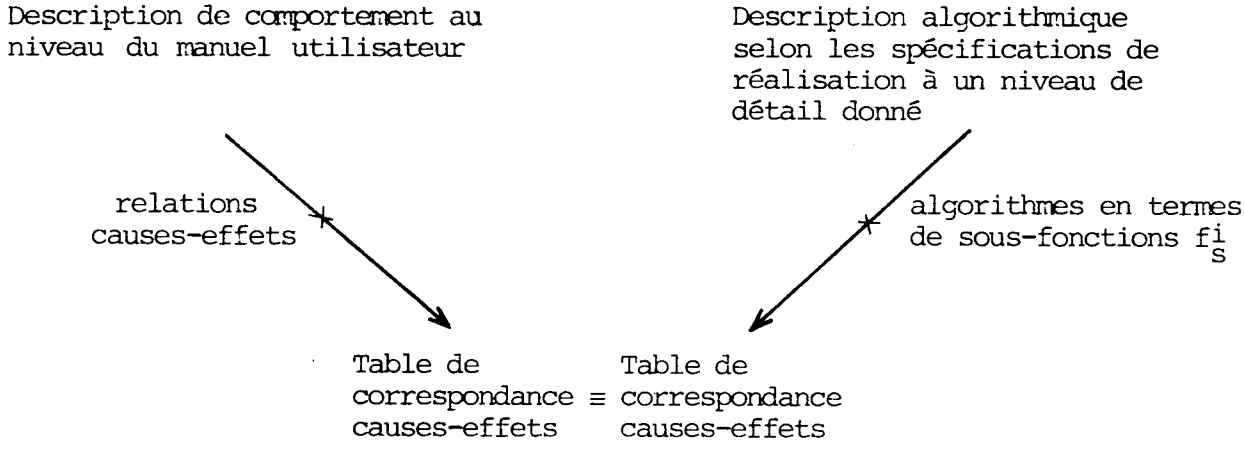
- Les *effets* correspondent aux sorties du système et aux transformations qui en résultent sur son état.

Une table de correspondance causes-effets est construite pour chaque fonction à partir des descriptions données dans les deux types de dossiers :

. Partant des manuels utilisateurs, cette construction est triviale;

. Partant des spécifications de réalisation, cette construction est obtenue en analysant les algorithmes décrivant l'implémentation des fonctions-système à partir de sous-fonctions f_s^i éléments de F^i (à un niveau de finesse donné).

i) Pour contrôler la cohérence des deux dossiers, il suffit de vérifier que ces deux tables se confondent.



ii) La sélection des configurations de test se fera par une couverture de l'une de ces tables de correspondance; ce qui revient à extraire un sous-ensemble de relations causes-effets dont la vérification équivaut "théoriquement" la vérification de toutes les relations causes-effets que traduit la table de correspondance. On expérimentera ensuite chaque relation causes-effets retenue sur un certain nombre d'exemples : chaque combinaison de causes détermine un ensemble de combinaisons de paramètres d'exécution parmi lesquelles on peut choisir un certain nombre de représentants.

La construction de la table de correspondance causes-effets à partir du manuel utilisateur est libre. La construction de cette table à partir des spécifications de réalisation est liée à la description donnée des algorithmes.

En pratique, tout le problème se résume à trouver le modèle adéquat qui permet le passage de la description donnée dans les spécifications de réalisation à la table de correspondance causes-effets. Il suffit ensuite de vérifier que la description donnée dans les manuels utilisateurs traduit exactement les relations causes-effets illustrées sur cette table.

Le modèle choisi est le graphe cause-effet que l'on va définir. Ce modèle superpose une description des algorithmes d'implémentation, à un niveau donné, à une description de comportement du type présenté dans les manuels utilisateurs. Cette description tient compte de la représentation du parallélisme et de la synchronisation entre les sous-fonctions qui composent les algorithmes. Elle permet aussi d'illustrer les contraintes, quand elles existent, entre les diverses causes qui déterminent le déroulement des fonctions-système. Enfin, le passage du modèle à la table de correspondance est trivial. Ces caractéristiques seront mises en évidence et justifiées au fur et à mesure.

II - 3. GRAPHE CAUSE-EFFET ASSOCIE A UNE FONCTION-SYSTEME

II - 3.1. Définition du graphe cause-effet

C'est un graphe orienté sans circuits (hiérarchie |D1|) comportant 5 types de noeuds :

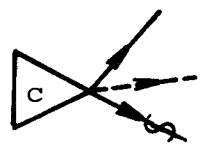
- des noeuds "cause-initiale" qui constituent des noeuds sources du graphe;
- des noeuds "effet-final" qui constituent des noeuds puits du graphe;
- des noeuds "cause-effet";
- des noeuds de contrôle;
- des noeuds de test.

Ces noeuds sont connectés entre eux par des arcs qui sont de deux types :

- des arcs "directs" que l'on représentera par une simple branche orientée. Exemple : N1 ⟶ N2;
- des arcs "complémentés" que l'on représentera par une branche orientée portant le symbole "∿". Exemple : N1 ⟶∿ N2.

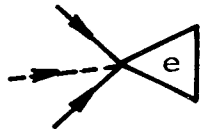
Description des noeuds

1) Un noeud cause-initiale est un noeud source qui peut avoir plusieurs arcs sortants des deux types. On le représentera par le symbole suivant :



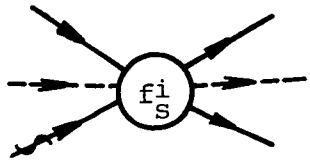
A un tel noeud est associée une condition logique "C" portant sur les valeurs d'un ou plusieurs des paramètres qui déterminent le déroulement de l'exécution de la fonction.

2) Un noeud effet final est un noeud puits qui peut avoir plusieurs arcs entrants, tous du type direct. On le représentera par le symbole

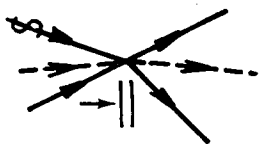


Un tel noeud porte une étiquette"e"qui représente un évènement pouvant résulter de l'exécution de la fonction (sorties du système et/ou transformations de son état).

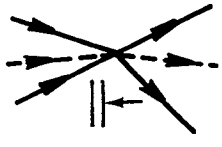
3) Un noeud cause-effet est un noeud que l'on représentera par un cercle portant une référence à une sous-fonction f_s^i de l'ensemble F^i. Un tel noeud a au moins un arc entrant et un arc sortant. Les arcs entrants peuvent être des deux types directs ou complémentés; les arcs sortants sont tous des arcs directs.



4) Un noeud de contrôle est un noeud que l'on représentera par un point portant l'une des deux étiquettes "→ ||" (convergence) et "|| ←" (divergence). Un tel noeud a au moins un arc entrant et un arc sortant. Quand le noeud porte l'étiquette "→ ||", les arcs entrants peuvent être des deux types direct ou complémenté, les arcs sortants sont tous du type direct. Quand le noeud porte l'étiquette "|| ←", les arcs entrants et sortants sont tous des arcs directs.



;



5) Un noeud de test est un noeud que l'on représentera par un point portant l'une des deux étiquettes "Λ" ou "V" symbolisant respectivement les deux opérateurs booléens ET et OU. Un tel noeud a toujours deux, et seulement deux, arcs entrants et au moins un arc sortant.

II - 3.2. Règles de construction du graphe cause-effet

La construction du graphe cause-effet doit respecter les règles suivantes :

r1 : Les successeurs directs d'un noeud de contrôle portant l'étiquette "→ ||" ne peuvent être que des noeuds cause-effet.

r2 : Les prédécesseurs directs d'un noeud de contrôle portant l'étiquette "|| ←" ne peuvent être que des noeuds cause-effet.

r3 : Les prédécesseurs directs d'un noeud effet-final ne peuvent être que des noeuds cause-effet ou cause-initiale.

La figure V-1 donne un exemple de graphe cause-effet.

II - 3.3. Interprétation logique des noeuds du graphe

Le graphe cause-effet peut être interprété comme un circuit logique combinatoire ayant en entrées les noeuds sources cause-initiale et en sorties les noeuds puits effet-final.

- Chaque noeud du graphe peut être dans l'un des deux états "actif " ou "inactif".
- Un arc direct est actif ssi le noeud à l'extrémité de départ est actif.
- Un arc complémenté est actif ssi le noeud à l'extrémité de départ est non actif.
- Un noeud cause-initiale est actif ssi la condition "c" qu'il porte est vérifiée.
- Un noeud cause-effet devient actif dès que l'un au moins des arcs entrants devient actif .
- Un noeud de contrôle portant l'étiquette "→ ||" devient actif dès que l'un au moins des arcs entrants devient actif.

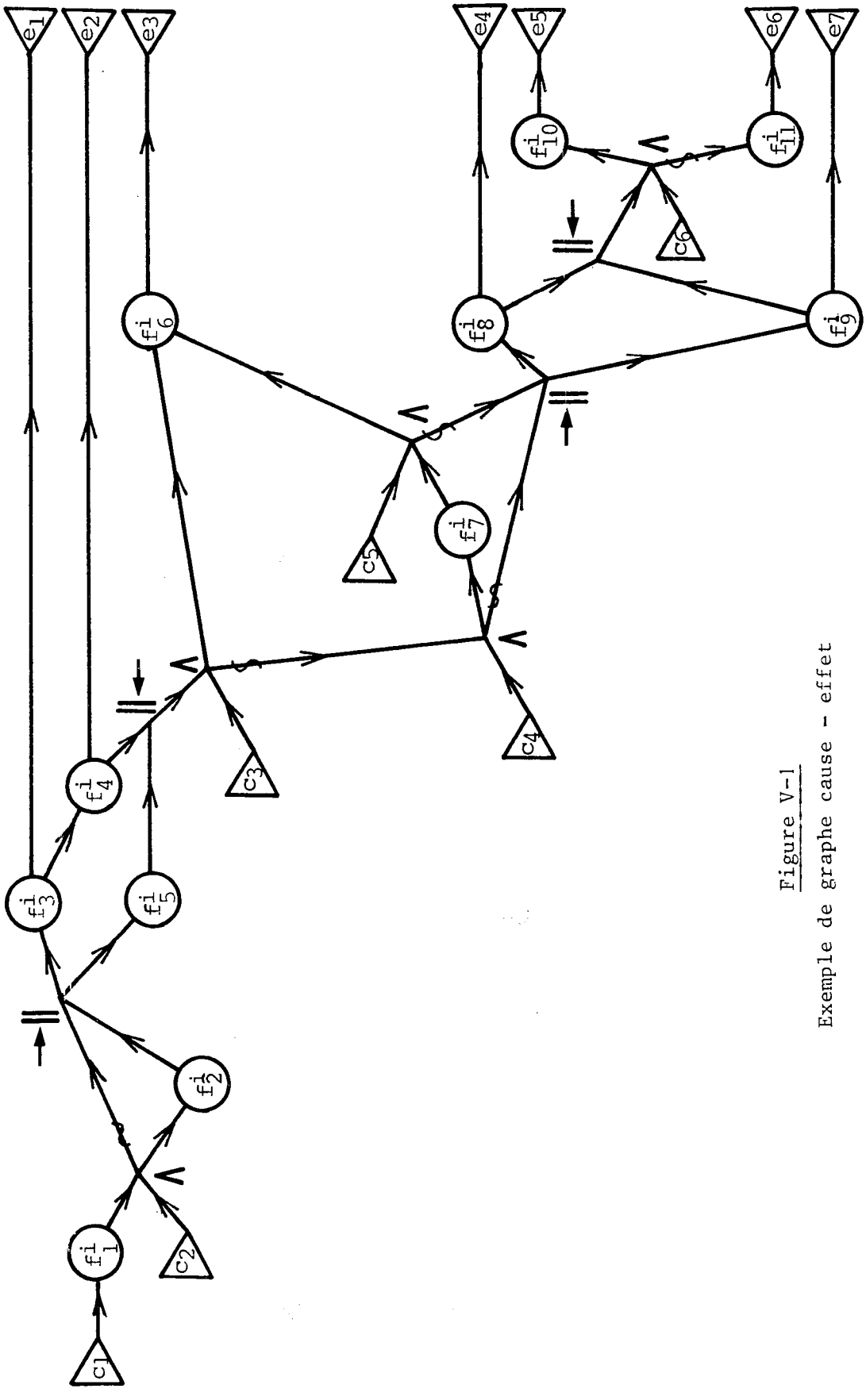


Figure V-1

Exemple de graphe cause - effet

- Un noeud de contrôle portant l'étiquette "|| ←" devient actif quand tous les arcs entrants deviennent actifs.

- Un noeud de test portant l'étiquette "Λ" est actif ssi tous les deux arcs entrants sont actifs.

- Un noeud de test portant l'étiquette "V" est actif ssi au moins l'un des deux arcs entrants est actif.

- Un noeud effet-final devient actif dès que l'un au moins des arcs entrants devient actif.

Définition

On dira qu'une configuration de causes-initiales est légale si l'interprétation logique du graphe pour cette configuration de causes n'amène aucune des situations suivantes :

- Deux ou plusieurs arcs entrants pour un même noeud de contrôle "→ ||" deviennent actifs;

- Deux ou plusieurs arcs entrants pour un même noeud cause-effet deviennent actifs.

II - 3.4. Graphe cause-effet associé à une fonction-système

Le graphe cause-effet tel qu'on vient de le définir n'est, en fait, qu'une représentation d'organigramme à laquelle on a apporté trois transformations essentielles.

Généralement, un organigramme classique comporte 4 types de noeuds :

. Un noeud de départ qui porte les valeurs des paramètres et données d'entrée;

. Des noeuds d'exécution qui effectuent des traitements sur les données;

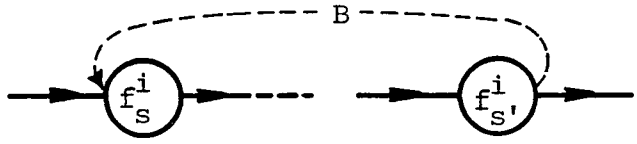
. Des noeuds de décision qui effectuent des tests sur les valeurs des paramètres et des données en cours de traitement;

. Un noeud final qui représente la sortie de l'organigramme.

1) On a, tout d'abord, introduit des noeuds de contrôle "→ ||" et "||←" qui permettent de représenter respectivement la divergence et la convergence de parallélisme.

2) Les noeuds d'exécution sont réduits aux sous-fonctions $f_s^i \in F^i$. Quand un noeud d'exécution effectue des transformations sur l'état des paramètres et des données d'entrée, les résultats de ces transformations sont portés dans un noeud effet-final connecté directement à la sortie de ce noeud.

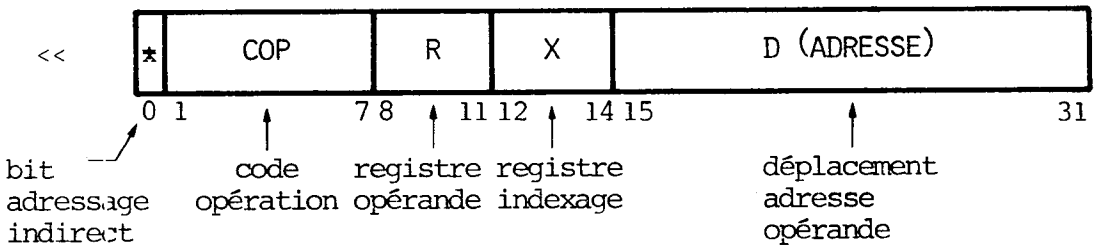
3) Les conditions de test aux noeuds de décision ne sont plus exprimées en fonction des valeurs des données au cours de leur transformation mais en fonction des valeurs initiales de départ. Chaque condition est décrite sous forme d'une expression booléenne sur des éléments causes-initiales et illustrée sur le graphe à l'aide des noeuds de test "V" et "Λ". Ceci est toujours possible si la description de la fonction ne comporte pas de boucles. Heureusement, de telles situations sont plutôt rares à un niveau adéquat de description des algorithmes des fonctions-système. Quand le cas se présente, on peut soit englober la boucle dans un seul noeud, soit ouvrir la boucle et représenter ses éléments une seule fois en indiquant en pointillé sur le graphe les éléments sur lesquels porte la boucle. Exemple :



En pratique, le problème de représentation des boucles se pose essentiellement pour la description des fonctions-système concernant les entrées/sorties et les opérations de traitement de chaines dans l'UC.

II - 3.5. Exemple : Graphe cause-effet associé à une fonction-instruction INTERPRET

Description de la fonction-instruction (telle qu'elle est donnée dans les manuels utilisateurs |A53|).



Cette instruction charge les bits 0 à 3 du mot effectif (ME) dans le code condition (CC); charge les bits 4 à 15 du mot effectif dans les bits 16 à 31 du registre R (et charge des zéros dans le reste du registre); enfin charge les bits 16 à 31 du mot effectif dans les bits 16 à 31 du registre Ru1 (et charge des zéros dans le reste du registre Ru1).

Sont modifiées : $[R]$, $[Rul]$ et CC

$[ME]_{0-3} \rightarrow CC$

$[ME]_{4-15} \rightarrow R_{20-31}$; $0 \rightarrow R_{0-19}$

$[ME]_{16-31} \rightarrow Rul_{16-31}$; $0 \rightarrow Rul_{0-15}$ >>

Graphe cause-effet associé (déduit du niveau le plus haut de description des spécifications de réalisation).

Ce graphe est celui donné en exemple en figure V-1 avec l'interprétation suivante des noeuds :

C_1 : lancement de l'exécution de l'instruction INTERPRET

C_2 : indexage demandé ($X \neq 0$)

C_3 : violation mémoire prévue sur premier accès pour recherche opérande

C_4 : bit adressage indirect positionné

C_5 : violation mémoire prévue sur deuxième accès pour recherche opérande

C_6 : R pair ($R \neq Rul$)

---

f_1^i : transcodage de l'instruction et calcul de l'adresse opérande ($[B] + D \rightarrow ADD$)

f_2^i : indexage ($ADD + [X] \rightarrow ADD$)

f_3^i : RAZ du registre R ($0 \rightarrow R$)

f_4^i : RAZ du registre Rul ($0 \rightarrow Rul$)

f_5^i : premier accès mémoire pour recherche opérande ($[ADD] \rightarrow ME$)

f_6^i : positionnement indicateurs violation mémoire

f_7^i : deuxième accès mémoire pour recherche opérande ($[ME] \rightarrow ME$)

f_8^i : positionnement code-condition ($[ME]_{0-3} \rightarrow CC$)

f_9^i : remplissage de R ($[ME]_{4-15} \rightarrow R_{20-31}$)

f_{10}^i : remplissage de Rul ($[ME]_{16-31} \rightarrow Rul_{16-31}$) (R pair)

f_{11}^i : remplissage de R ($[ME]_{16-31} \rightarrow R_{16-31}$) (R impair)

---

e_1 : $R = 0$

e_2 : $Rul = 0$

e_3 : déroutement pour violation mémoire

e_4 : $CC = ME_{0-3}$

e_5 : $Rul_{16-31} = ME_{16-31}$, $Rul_{0-15} = 0$

e_6 : $R_{16-31} = ME_{16-31}$, $R_{0-15} = 0$

e_7 : $R_{20-31} = ME_{4-15}$, $R_{0-19} = 0$

II - 3.6. Contraintes de commande et d'observation entre les noeuds du graphe

Pour que la description des fonctions-système sous forme de graphe cause-effet soit complète, il faut qu'elle tienne compte des contraintes de commande et d'observation entre les noeuds du graphe. Ces contraintes sont imposées par les moyens d'activation et de contrôle du système et peuvent se présenter sous l'une des formes suivantes :

- Deux ou plusieurs causes sont mutuellement exclusives, c'est-à-dire qu'une au plus de ces causes peut être impliquée dans l'activation de la fonction-système;

- Une au moins parmi un ensemble de causes est obligatoire dans toute configuration d'activation de la fonction-système;

- Une et une seule parmi un ensemble de causes doit être impliquée dans toute configuration d'activation de la fonction-système;

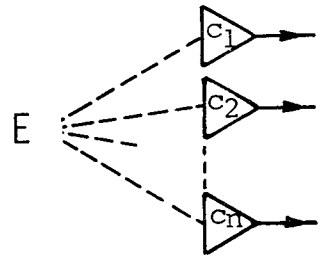
- L'implication d'une cause requiert l'implication d'une ou de plusieurs autres causes;

- La production d'un effet masque l'observation d'autres effets.

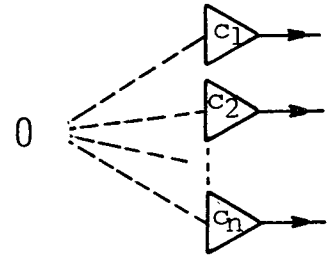
Dans l'exemple présenté ci-dessus, l'implication de la cause c_5 requiert l'implication de la cause c_4. Aussi, la production des effets e_5 et e_7 masque l'observation des effets e_2 et e_1; l'observation des effets e_1 et e_2 n'est possible qu'en cas de fin sur déroutement (effet e_3) résultant de la vérification de l'une des deux causes c_3 et c_5.

Les exemples donnés en figure V-2 montrent une façon de représenter ces contraintes sur le graphe.

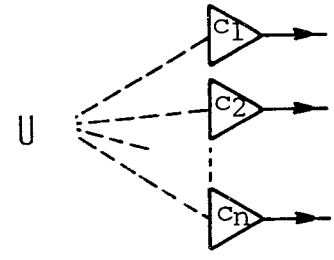
1 - Les causes c_1, c_2, ... c_n sont mutuellement exclusives



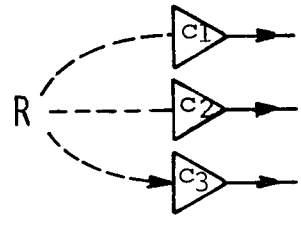
2 - Une au moins des causes c_1, c_2, ... c_n est obligatoire



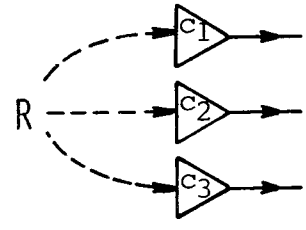
3 - Une et une seule parmi les causes c_1, c_2, ... c_n doit être impliquée dans toute activation de la fonction



4 - Les causes c_1 et c_2 ne peuvent être impliquées sans la cause c_3



5 - Les causes c_1, c_2 et c_3 ne peuvent être impliquées séparément



6 - La production de l'effet e_1 masque l'observation des effets e_2 et e_3

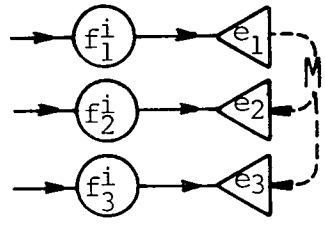


Figure V-2

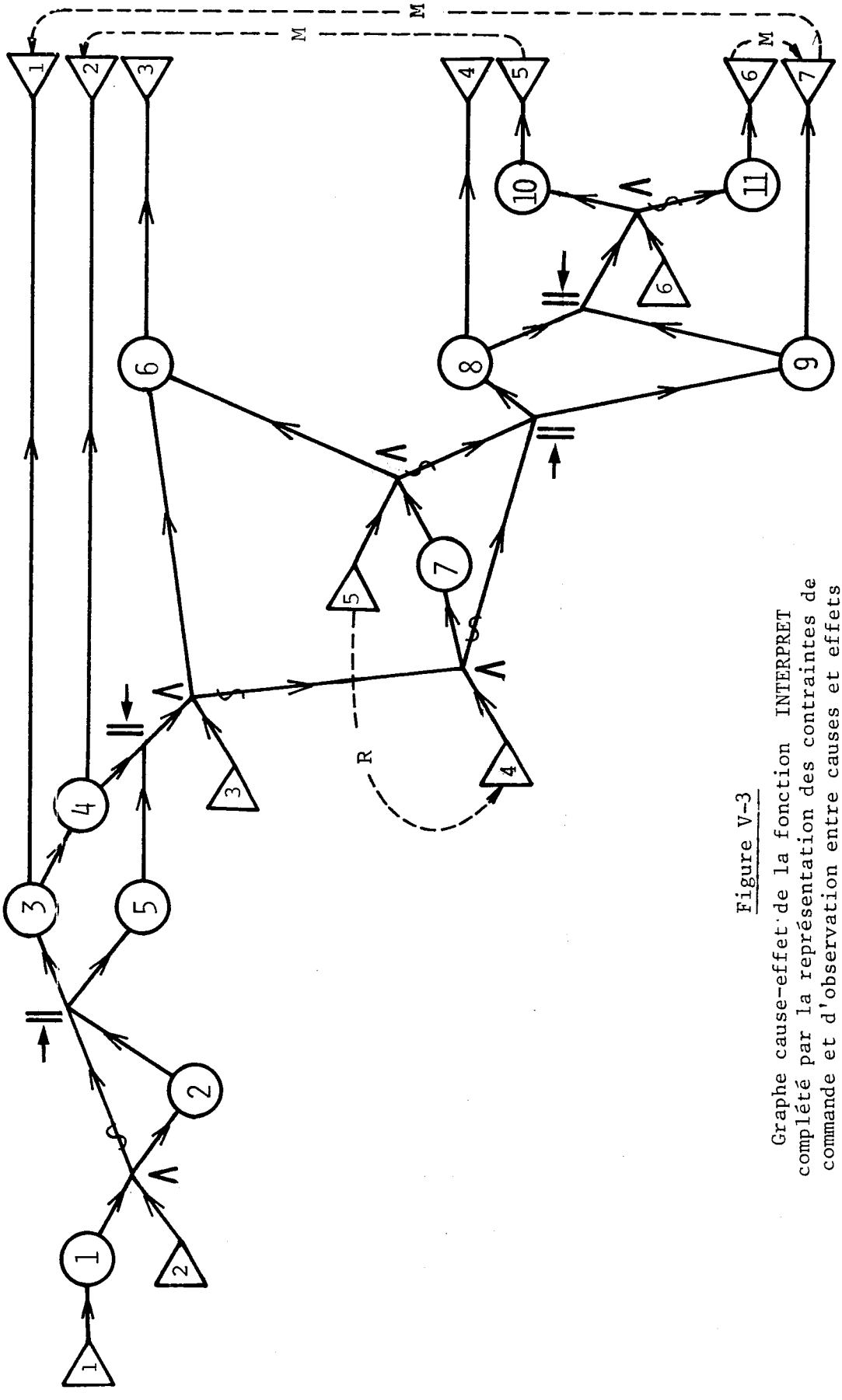


Figure V-3

Graphe cause-effet de la fonction INTERPRET complété par la représentation des contraintes de commande et d'observation entre causes et effets

III - METHODES ET STRATEGIES DE TEST

- Etant donné un graphe cause-effet associé à une fonction-système, on dira qu'une configuration de causes-initiales est compatible si elle satisfait les contraintes de commande illustrées sur le graphe.

Dans cette configuration, une cause-initiale peut être :

. impliquée (I);

. supprimée (S);

. ou considérée dans un état indifférent (X).

Si l'on observe la configuration correspondante des effets-finaux, un effet peut être :

. présent (P);

. absent (A);

. ou masqué (M).

- Pour un graphe correctement construit, toute configuration de causes compatible est forcément légale (Def. § II-3.3).

La table de correspondance cause-effet (dont il a été question dans le paragraphe II-2) peut être obtenue directement à partir du graphe cause-effet. Si k est le nombre de noeuds cause-initiale du graphe, et ℓ le nombre de noeuds effet-final, cette table aura $k+\ell$ colonnes et autant de lignes qu'il existe de configurations de causes compatibles qui déterminent un chemin d'exécution distinct de la fonction. Le remplissage de cette table se fera de la manière suivante :

. On associe les k premières colonnes aux k causes-initiales, une colonne pour chaque cause-initiale, et les ℓ vecteurs colonnes qui restent aux ℓ effets-finaux, une colonne pour chaque effet-final.

. Pour chaque configuration de causes-initiales qui est retenue, on associe un vecteur ligne.

. Sur ce vecteur, on marque les k premières cases des lettres I, S ou X respectivement selon que la cause-initiale correspondante est considérée comme impliquée, supprimée ou indifférente. On marque les ℓ cases qui suivent des lettres P, A ou M respectivement selon que dans la configuration des effets-finaux correspondante, l'effet-final est présent, absent ou masqué.

. Chaque vecteur ligne de la table constitue ce que l'on appellera dans la suite un cas-de-test.

La figure IV-4 illustre cette table pour l'exemple donné en figure V.2.

c_1	c_2	c_3	c_4	c_5	c_6	e_1	e_2	e_3	e_4	e_5	e_6	e_7
I	I	I	X	X	X	P	P	P	A	A	A	A
I	S	I	X	X	X	P	P	P	A	A	A	A
I	I	S	I	I	X	P	P	P	A	A	A	A
I	S	S	I	I	X	P	P	P	A	A	A	A
I	I	S	I	S	I	M	M	A	P	P	A	P
I	S	S	S	X	I	M	M	A	P	P	A	P
I	I	S	I	S	I	M	M	A	P	P	A	P
I	S	S	S	X	I	M	M	A	P	P	A	P
I	I	S	I	S	S	M	M	A	P	A	P	M
I	S	S	S	X	S	M	M	A	P	A	P	M
I	I	S	I	S	S	M	M	A	P	A	P	M
I	S	S	S	X	S	M	M	A	P	A	P	M

Figure IV-4

La couverture de la table pour la recherche des configurations de causes intéressantes pour le test peut être faite selon trois degrés d'exhaustivité :

1) Faire figurer tout arc sortant d'un noeud de test sur au moins l'un des chemins d'exécution correspondant aux cas-de-test retenus; ce qui revient, en pratique, à garantir que toute décision dans l'algorithme d'exécution de la fonction-système et, à fortiori, toute sous-fonction f_s^i impliquée dans cet algorithme, sera expérimentée un nombre minimum de fois.

2) Chercher des recoupements des cas-de-tests choisis permettant de localiser au mieux le noeud respondable en cas de détection de désaccord entre les résultats obtenus et ceux attendus.

3) Considérer tous les cas-de-tests possibles.

S'il s'agissait véritablement du test d'un circuit logique, on chercherait un ensemble minimal de cas-de-tests qui puisse satisfaire l'un des deux critères 1 ou 2 selon que l'on s'intéresse ou non à la localisation; l'élaboration des tests selon le critère 3 aurait introduit une redondance inutile. Cela tient au fait qu'un circuit logique représente le flot d'information en général (contrôle et données), et donne la trace précise des chemins d'exécution pour toute configuration d'entrées. Le test de chaque noeud du circuit peut être fait indifféremment en sensibilisant l'un quelconque des chemins d'exécution qui l'utilisent. Le graphe cause-effet n'est qu'un organigramme qui représente seulement le flot de contrôle sans précision de la trace du flot de données. La façon dont est activé chaque noeud (en particulier les noeuds cause-effet) peut dépendre du chemin d'exécution qui utilise ce noeud. Pour garantir l'exhaustivité souhaitable, il semble de ce fait préférable de définir les tests selon le critère 3.

IV - EFFICACITE

Le graphe cause-effet fait le joint entre une description de comportement du niveau des manuels utilisateurs et une description fonctionnelle qui illustre, à un niveau de détail donné, les algorithmes qui ont été adoptés dans l'implémentation matérielle des fonctions-système. Ce modèle se veut être, surtout, un outil d'aide à la compréhension du fonctionnement du système. Pour cela, on a tenu à rester au plus près d'une représentation d'organigramme et à distinguer la signification fonctionnelle de chaque noeud (bien que certains de ces noeuds aient la même interprétation logique), afin que cette représentation soit la plus parlante et mieux adaptée à la description des algorithmes. La vérification de la "légalité" des configurations de causes compatibles est un moyen de s'assurer de la construction correcte du graphe cause-effet.

Le fait de rester proche des algorithmes d'implémentation du matériel (et en particulier de relater le parallélisme) permet de mieux vérifier que la description du rapport cause-effet donnée dans le manuel utilisateur est complète. Dans l'exemple qui a été proposé en V-4.5, la description de la fonction-instruction INTERPRET extraite du manuel utilisateur est incomplète puisqu'elle ne permet pas de savoir l'état final des registres R et Rul en cas de déroutement pour violation mémoire. L'utilisateur peut considérer que

le contenu de ces registres est inchangé; or, l'examen du graphe cause-effet montre, qu'en cas de déroutement pour violation mémoire, ces registres se trouvent remis à zéro. Il en aurait été autrement si les sous-fonctions f_5^i, f_3^i et f_4^i étaient exécutées en séquence plutôt qu'en parallèle. Seule donc une représentation à partir des algorithmes d'implémentation permet de rendre compte de ce fait.

La connaissance aussi détaillée que possible du rapport cause-effet permet en outre de mieux exploiter les possibilités d'observation du comportement du système pour des fins de localisation des fautes.

Les critères invoqués pour la génération des tests correspondent à des critères de recouvrement d'algorithme [D3] . Ils garantissent l'activation, pour toute fonction-système, de chaque élément de décision et chaque sous-fonction utilisée. Le test selon les choix de données manipulées par les sous-fonctions est une préoccupation secondaire que l'on peut résoudre de façon déterministe ou aléatoire en choisissant les paramètres et données d'entrée dans les limites imposées par les contraintes de test.

Cette technique ne permet pas de garantir la conformité absolue du système. Néanmoins, elle tend vers une vérification fonctionnelle systématique. Elle permet de couvrir toutes les situations d'activation possibles des fonctions du système, en tenant compte au mieux de la façon dont le système réalise ces fonctions. De plus, elle apparaît comme le seul moyen d'obtenir un diagnostic fonctionnel localisant une faute.

Le degré de résolution du diagnostic dépend naturellement du niveau de finesse considéré dans la description de l'algorithme; il convient de trouver un compromis qui permet de maintenir une certaine facilité pour la génération des tests.

CHAPITRE VI

OUTIL LOGICIEL SUPPORT DES TESTS D'ACCEPTATION EN INTEGRATION DES SYSTEMES. LE PROGRAMME SIMUX*1

* 1. Pour test de SImultanéité et de MUltiplexages des machines de la gamme X.

Le programme SIMUX |D7| a été développé dans le cadre d'une collaboration CII[*2], *IRIA* [*3] *et ENSIMAG* [*4] *pour les tests d'intégration sur plateformes des systèmes de la nouvelle gamme X (UNIDATA 7740-60 |A53|). La réalisation technique de ce programme a été menée entièrement par les services DFQ/TP de la CII. Ce chapitre reprend les termes d'une spécification de définition, proposée par l'auteur, dans le cadre de sa participation au développement de ce produit. Le but de cette spécification est de suggérer des idées concernant l'organisation d'un tel programme plutôt que de décrire des solutions techniques.*

*2 - CII - Département Fiabilité et Qualité/TP

*3 - IRIA-LABORIA - Equipe CETASI, Groupe des Projets sur le Temps Réel et les Réseaux

*4 : ENSIMAG - Equipe Conception et Sécurité des Systèmes Logiques.

I - OBJECTIFS

Il s'agit de pourvoir un support logiciel qui permet de gérer en fonctionnement autonome (stand-alone) l'activation de processus, dans le but d'effectuer les tests d'acceptation en intégration des systèmes sur plateformes.

Les objectifs recherchés sont les suivants :

- Vérifier la bonne intégration des modules du système, et ce avec un minimum d'intervention humaine.

- Automatiser la procédure de diagnostic en cas de détection d'anomalies.

- Pourvoir des moyens de mesure qui rendent possible le contrôle des performances spécifiques à chaque module et, éventuellement, la mise en évidence à travers ces mesures d'une non-conformité aux clauses des spécifications fonctionnelles qui définissent le comportement de ces modules.

- Effectuer la phase de déverminage dans un environnement semblable à celui d'une exploitation réelle.

- Aider, par l'élaboration automatique d'un "fichier historique de contextes d'erreurs", la collecte de données statistiques qui intéressent les services d'études afin d'améliorer la qualité de service et de fiabilité de nouveaux produits.

II - PRESENTATION BREVE DE L'ARCHITECTURE DU SYSTEME UNIDATA-7760

La figure VI-1 présente l'architecture du système UNIDATA 7760.

Description des unités mémoire

L'unité mémoire est la portion physiquement reconfigurable de la mémoire principale. Elle est constituée de 4 blocs de capacité maximum 256 K-octets ou 1 M-octets (Figure VI-2). Chaque bloc est organisé en mots de 72 bits dont 64 bits d'information utile. Les blocs 0 et 1 d'une part, et les blocs 2 et 3 d'autre part, sont associés pour former des Demi-Unités Mémoire (DUM) qui travaillent en synchronisme permettant d'avoir un mot mémoire de 128 bits utiles.

Les blocs sont connectés à la Logique d'Unité Mémoire (LUM) par l'intermédiaire d'un Interface Standard de Bloc (ISB).

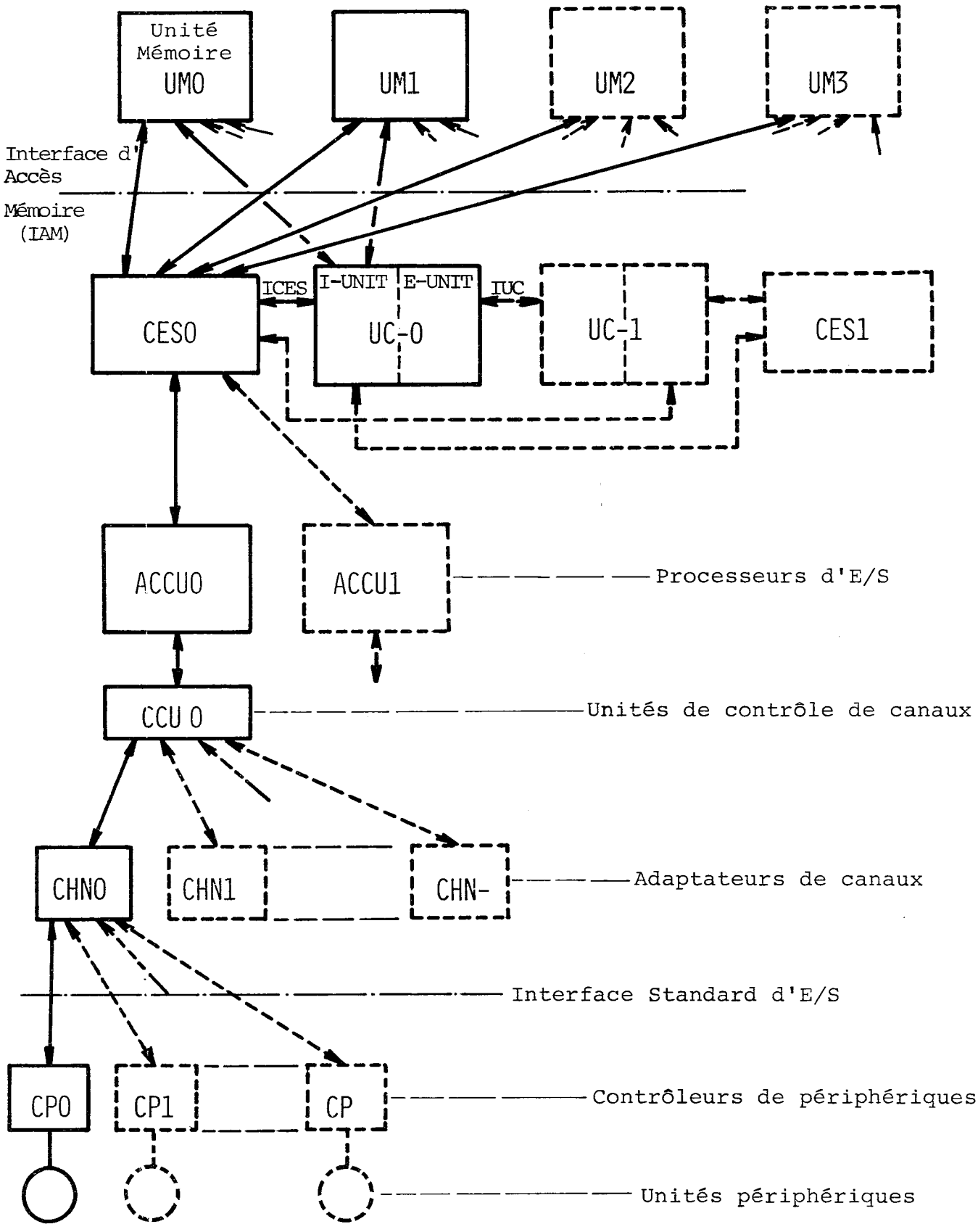


Figure VI-1

Architecture du système UNIDATA-7760

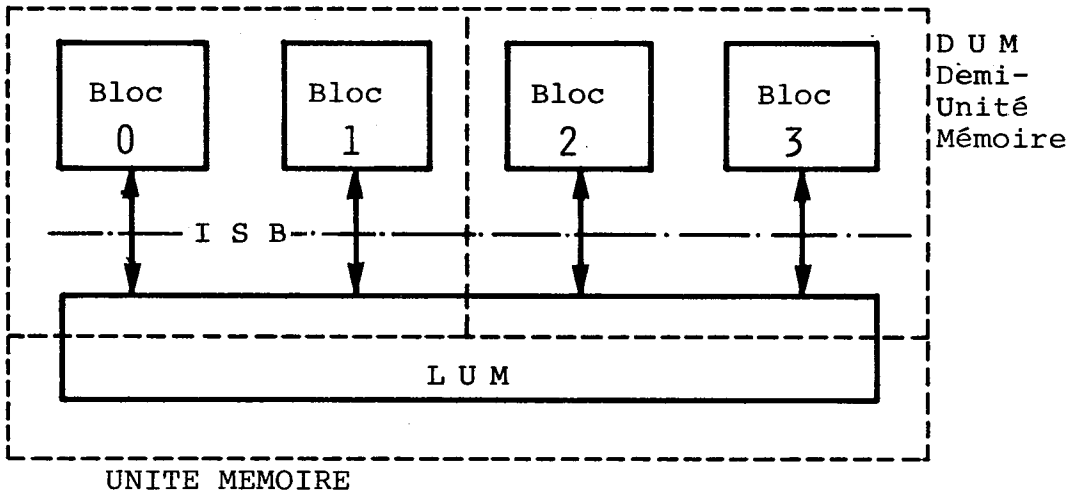


Figure VI-2

Description des unités de calcul

Le système peut comporter une à deux unités de calcul. Chaque unité est composée de deux sous-systèmes microprogrammés, indépendants au point de vue fonctionnel mais synchronisés sur la même horloge au point de vue du traitement. Ces deux sous-systèmes correspondent à l'unité d'instruction et l'unité d'exécution (Figure VI-3).

L'unité d'instruction se compose de 3 parties :

1) La logique de préparation (STPR) chargée de l'appel des instructions, du calcul de l'adresse, de la gestion de l'état programme et du traitement des instructions de branchement et des instructions privilégiées.

2) La logique de prémémoire (STPM) constituée d'une antémémoire et de la logique associée chargée de la gestion et de l'anticipation.

3) La logique d'interface (STIN) qui se divise en deux parties :

- . La logique d'interface avec les unités mémoire, commandée par la STPM;
- . La logique d'interface avec les autres unités extérieures (UC,CES..) commandée par la STPR.

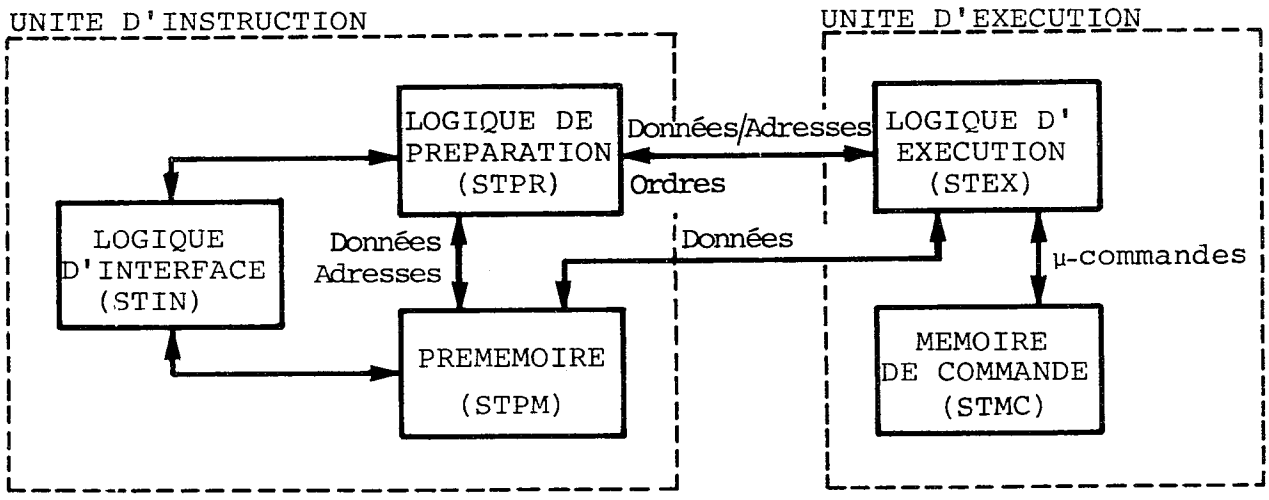


Figure VI-3

L'unité d'exécution se compose de deux parties :

1) La logique d'exécution (STEX) qui traite les instructions non-privilégiées.

2) La mémoire de commande (STMC) qui contient les microprogrammes de la STEX.

L'unité de calcul permet l'exécution de quatre instructions simultanément, à différents stades de leur exécution (technique du pipe-lining), grâce à des dispositifs d'anticipation.

Description du système d'entrées/sorties

On distingue 6 types de modules qui se situent à 6 niveaux hiérarchisés :

1) Le multiplexeur d'E/S ou CES (Contrôleur d'E/S). Ce module a pour fonction d'assurer l'interface du système d'entrées/sorties avec les unités de calcul et les unités mémoire. Il peut connecter un ou deux processeurs d'entrées/sorties.

2) Le processeur d'E/S ou ACCU (Adaptateur d'Unité de Contrôle de Canaux). Processeur entièrement microprogrammé, qui assure les traitements correspondant aux opérations d'entrées/sorties pour un groupe de canaux connectés à une même unité de contrôle de canaux (CCU).

3) L'unité de contrôle des canaux (CCU). Elle constitue une logique commune à un groupe de canaux pour qui elle assure le multiplexage d'information venant de (ou allant vers) la mémoire principale à travers l'ACCU, et gère les priorités respectives de leur interactions avec l'ACCU.

4) Les modules canaux. Ils permettent de connecter un ou plusieurs contrôleurs de périphériques sur des interfaces standards d'E/S. On en distingue deux types :

- Les canaux multiplexés par octet (BYMUX). Ils permettent la connexion de périphériques lents travaillant en transfert par octet tels que lecteurs de cartes et téléscriptrices. Les contrôleurs de périphériques qui leur sont connectés peuvent être desservis simultanément sur la base d'un multiplexage par octet.

- Les canaux multiplexés par bloc (BLMUX). Ils permettent la connexion de périphériques rapides tels que des unités bande magnétique ou des unités disque. Un BLMUX peut travailler en deux modes :

. En mode "multiplexé", il réalise des transferts de données sur la base d'un multiplexage par bloc dont la longueur dépend du programme canal et des caractéristiques de l'unité périphérique ;

. En mode "sélecteur", il exécute une seule opération d'entrée/sortie à la fois. Une deuxième opération ne peut être acceptée qu'après achèvement de l'opération en cours.

5) Les contrôleurs de périphériques (CP). Ils sont de quatre types :

. CP mono-appareil (ne peut connecter qu'un seul périphérique).

. CP multi-appareils non multiplexé (peut connecter plusieurs appareils mais travaillant avec un seul à la fois).

. CP multi-appareils multiplexé par octet.

. CP multi-appareils multiplexé par bloc.

6) Au dernier niveau, on trouve les unités périphériques. Les modules CES, CCU et canaux sont pilotés par le processeur ACCU. Chacun de ces modules est doté d'une logique de commande qui, chaque fois que le module est sollicité, doit faire appel à l'assistance d'un microprogramme exécuté dans l'ACCU. (Excepté le cas du BLMUX qui est doté de deux logiques de commande dont une est entièrement câblée).

III - CONSIDERATIONS DE TEST

L'exécution d'une fonction du système utilise un certain nombre de modules. Mais, la configuration du système peut comporter plusieurs modules identiques, si bien que la fonction peut être exécutée sur plusieurs sous-configurations différentes.

Une première étude à l'aide de descriptions sous forme de graphe cause-effet permet de déterminer, pour toute sous-configuration type, quels sont les cas-de-tests intéressants qu'il faut expérimenter pour chaque fonction pouvant être exécutée par cette sous-configuration.

Une deuxième étude permet de déterminer, pour toute configuration possible du système, quelles sont les sous-configurations équivalentes pour lesquelles il faut appliquer les mêmes tests.

Dans la pratique, cela peut être ramené :

- D'une part, à élaborer un ensemble de "segments de code", chaque segment de code concrétise les cas-de-tests pour les fonctions exécutées par une sous-configuration type. Les adresses identifiant les modules constituant la sous-configuration sont considérées comme des paramètres.

- D'autre part, à élaborer un générateur de programmes de test qui, à partir de ces segments de code et des données définissant la configuration d'un système à tester, génère les programmes de test à faire exécuter par ce système (Figure VI-4).

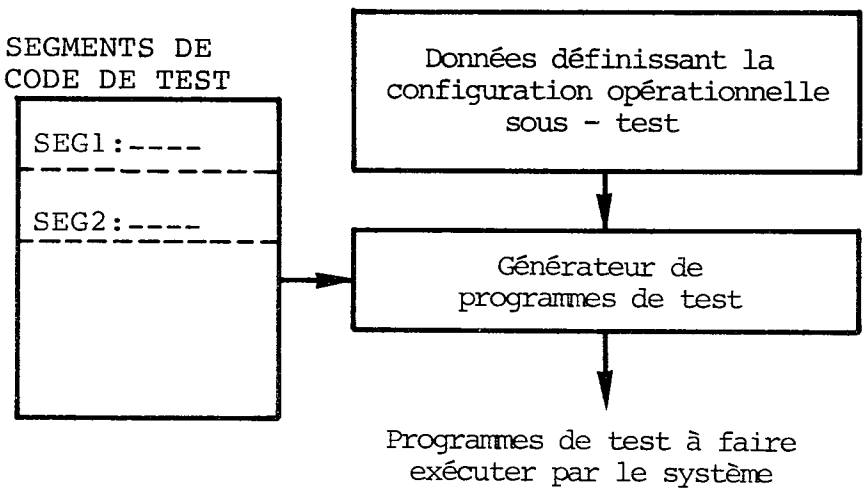


Figure VI-4

SITUATIONS D'ACTIVITE DE SIMULTANEITE ET DE CHARGE

L'implémentation des fonctions du système donne la possibilité d'exécuter un certain nombre de ces fonctions en parallèle. Dans ce fonctionnement, un module du système peut être amené à répondre à des demandes de service simultanées émanant des modules qui lui sont connectés.

Une particularité importante concerne les demandes qui doivent impérativement être satisfaites dans des délais déterminés sous peine de forcer la fin anormale d'une opération en cours. De telles demandes génèrent une situation d'activité critique au sein du module si, au moment de leur arrivée, le module se trouve occupé à satisfaire des demandes précédentes (éventuellement plus prioritaires), ou si la satisfaction de ces demandes nécessite la participation d'un module tiers qui peut, lui-même, se trouver occupé avec d'autres demandes.

Les tests d'acceptation doivent vérifier l'harmonie des caractéristiques de fonctionnement et de performances des modules du système dans l'accomplissement en parallèle de ces fonctions. Ce qui amène à générer des situations d'activité critique au sein de ces modules et à vérifier que ces modules supportent correctement de telles situations.

ETUDE D'UN EXEMPLE TYPE : Problème de l'erreur de rythme lors d'une opération d'entrée/sortie

Lors d'une opération de lecture ou d'écriture sur la plupart des sous-systèmes périphériques, le transfert de l'information, une fois initialisé entre le périphérique et le contrôleur, s'effectue à une cadence constante. Le long d'un transfert en entrée, le contrôleur doit disposer d'un espace tampon libre pour pouvoir y ranger les informations venant du périphérique. En cas d'un transfert en sortie, le contrôleur doit disposer de l'information prête pour pouvoir alimenter à cadence normale le périphérique concerné par le transfert.

Cela amène le contrôleur à effectuer des séquences de transfert élémentaire avec le canal afin de libérer (en entrée) ou remplir (en sortie) une partie de son tampon. Une erreur de rythme se produit quand le périphérique demande à entrer de l'information alors que le tampon du contrôleur se

trouve saturé, ou quand le périphérique demande de l'information en sortie, alors que le tampon se trouve vide. Dans les deux cas, il y a perte d'information et abandon sur fin anormale de l'opération en cours.

L'état d'activité du contrôleur va donc dépendre de l'état d'occupation du canal qui peut ou non prendre en compte de suite les demandes de transfert élémentaire formulées par le contrôleur. Dans le cas d'un contrôleur multi-appareils, l'état d'activité critique de celui-ci dépendra aussi du nombre de périphériques connectés à ce contrôleur, qui ont été activés en parallèle pour des opérations de transfert.

Un raisonnement analogue peut être reconduit pour le canal. Les temps de réponse du canal pour des demandes de transfert émises par le contrôleur dépendent, d'une part, de la charge du canal exprimée par le nombre de contrôleurs initialisés pour des échanges simultanés, d'autre part, de l'état d'occupation du processeur ACCU.

Un fait à souligner est que le déroulement d'une opération d'entrée/sortie pour une même longueur de transfert et une même voie d'E/S est différent selon la manière dont cette opération est programmée. Le transfert de n octets d'information peut être programmé en une seule commande-canal ou en plusieurs commandes-canal chaînées en données.

Dans le deuxième cas, il y a un traitement supplémentaire pour chaque chainage rencontré, qui consiste pour le canal à rentrer en mémoire les octets d'information lus pour le compte de la commande en cours et à aller chercher en mémoire la prochaine commande à traiter. Ces accès mémoire supplémentaires sont de nature à accentuer une situation d'activité critique au sein du canal, puisque pendant ce temps le contrôleur continue à demander à effectuer des transferts élémentaires.

L'examen détaillé de la logique de fonctionnement du canal dégage un certain nombre de facteurs qui favorisent la génération d'une situation d'activité critique.

1) Le nombre de contrôleurs devant être desservis simultanément par le canal est important;

2) Les transferts sont programmés en plusieurs commandes-canal en chainage de données;

3) Le procédé de chainage en fin de page mémoire physique est utilisé;

4) Les transferts simultanés s'effectuent dans le même bloc mémoire;

5) Les blocs mémoire où s'effectuent les transferts sont accédés simultanément par l'ensemble des UCs et CESs présents;

6) Le CES supportant le canal est en charge maximale avec les adaptateurs qui lui sont connectés;

7) Le processeur ACCU supportant le canal est en charge maximale avec les autres canaux qu'il regroupe sous le même CCU;

8) Le processeur ACCU est appelé à répondre à des demandes UC fréquentes concernant le test de l'état du canal ou le test de l'état du périphérique;

9) Les zones sous-canaux sont dans le même bloc mémoire où s'effectuent les transferts.

La plupart de ces situations sont, généralement, judicieusement évitées par les concepteurs du logiciel et d'exploitation. Il n'empêche que des situations du type 1, 3, 5, 6 et 7 peuvent se produire et favoriser la génération d'un fonctionnement critique au sein des modules du système.

Il reste à tirer de cet exemple le renseignement suivant : une situation d'activité critique d'un module peut être due soit à la saturation de l'environnement du module, soit à la saturation du module même, et à fortiori au cumul des deux situations.

Pour le cas du module canal, les facteurs 5, 6, 7 et 8 sont de nature à favoriser la saturation de son environnement. Les facteurs 1, 2, 3, 4 et 9 sont de nature à saturer le canal.

THEMES QUE L'ON PEUT RETENIR POUR L'ELABORATION DES PROGRAMMES DE TEST DE SIMULTANEITE ET DE CHARGE

T1. Multiplexage des transferts dans les contrôleurs multi-appareils multiplexés.

T2. Activité de simultanéité interne en mode standard (sans saturation) dans les canaux.

T3. Activité de simultanéité critique dans les canaux.

T4. Simultanéité inter-canaux, pour les canaux connectés à un même CCU. Multiplexage des phases dans un ACCU.

T5. Simultanéité inter-ACCU pour un même CES.

T6. Accès simultanés sur une même demi-unité mémoire (DUM).

T7. Accès simultanés sur les deux DUM d'une même unité mémoire.

T8. Simultanéité entre UC et CESs.

T9. Simultanéité inter-UCs.

T10. Simultanéité globale UCs et CESs.

IV - LES PROGRAMMES DE TEST

IV - 1. PROTEST ET TACHE DE TEST

La séquence de code qui concrétise un cas-de-test pour une fonction donnée se compose de trois sous-séquences :

- Une sous-séquence *d'initialisation* qui prépare le contexte correspondant à cette situation d'exécution : les causes initiales correspondant à une situation d'exécution peuvent porter sur la valeur du code-condition, ou sur le contenu de certains registres ou mots mémoire... Il faut préparer cette information avant le lancement de l'exécution de la fonction.

- Une sous-séquence *d'exécution* qui correspond à l'activation de la fonction pour la situation d'exécution testée.

- Une sous-séquence de *vérification* qui analyse les résultats obtenus : comparaison des résultats obtenus avec ceux attendus et déduction d'un diagnostic.

On appellera une telle séquence un *protest*[*1].

On peut distinguer deux classes de protests :

- La classe des protests UC;
- La classe des protests d'E/S

Classe des protests UC

Elle regroupe trois familles de protests :

1) Une première famille concerne la vérification des instructions de branchement et des instructions privilégiées autres que les instructions

*1. On a déjà utilisé cette dénomination pour la même signification dans le chapitre IV.

d'E/S.Ces instructions sont entièrement exécutées dans la station de préparation de l'UC. Cette famille de protests concerne aussi le test des horloges, du dispositif de translation d'adresse et de gestion de l'antémémoire.

2) Une deuxième famille concerne la vérification des instructions de chargement, de rangement et de transfert en mémoire avec prise en compte des déroutements. Les blocs mémoire accédés par ces protests peuvent être initialisés à l'avance par chargement via un point d'accès mémoire autre que celui utilisé par l'UC (exemple : chargement direct à partir d'une unité périphérique).

3) Une troisième famille concerne la vérification des instructions non-privilégiées avec prise en compte des déroutements. Les vérifications des instructions de calcul peuvent être effectuées :

. soit par déroulements déterministes (en GO-NO-GO) : choix déterministe des profils de données et comparaison à des résultats connus.

. soit par déroulements aléatoires (FILS) : génération aléatoire de profils de données et vérification par procédure logicielle.

Classe des protests d'E/S

Les protests de cette classe concernent la vérification des fonctions réalisées par les divers modules constituant le système d'entrées/sorties.

Un exemple simple d'un protest d'E/S peut être le suivant :

- Protest d'écriture d'un secteur sur une unité disque magnétique.

-- But : contrôler le déroulement des phases d'une opération d'écriture avec compte d'octets égal à un secteur.

sous-séquence d'initialisation :

. Préparation en mémoire d'un tampon d'écriture avec un code approprié;

. Préparation d'un programme canal de positionnement;

. Préparation d'un programme canal de recherche et d'écriture d'un secteur;

. Remise à zéro des registres d'état du périphérique sous test;

. Exécution du programme canal de positionnement;

. Traitement des réponses attendues sur exécution des instructions de test d'état périphérique et test d'état canal;

sous-séquence d'exécution :

. Lancement de l'exécution du programme canal d'écriture;

. Vérification du code-condition rendu et mise en attente sur montée d'interruption (IT) ou déclenchement d'une horloge de garde;

sous-séquence de vérification :

. Acquittement de l'IT et traitement des réponses attendues sur exécution des instructions de test d'état périphérique et test d'état canal;

. Vérification de la bonne écriture, éventuellement, par exécution d'un ordre contrôle d'écriture ou par lecture et comparaison au tampon mémoire ayant servi à l'écriture.

Une *tâche de test* est un ensemble de protests regroupés pour l'un des deux objectifs suivants :

i) Pour être exécutés en séquence de manière à effectuer un test complet d'une fonction dans les diverses situations d'exécution sans parallélisme. On parle alors de *tâche de test unitaire.*

ii) Pour être exécutés en parallèle de manière à générer une situation d'activité de simultanéité au sein d'un module. On parle de *tâche de test de simultanéité.*

Une tâche de test de simultanéité peut être élaborée :

α) Soit pour générer une situation d'activité de simultanéité en mode standard au sein d'un module : les contextes d'exécution des protests sont choisis de façon à favoriser l'accomplissement correct en simultanéité des opérations testées.

Exemple : Pour le cas d'un canal connecté à trois contrôleurs commandant chacun une unité disque, une tâche de test de simultanéité en mode standard peut consister à activer en parallèle trois opérations de lecture d'un secteur, une avec chaque contrôleur, sans chainage de données et avec des adresses mémoire situées dans des blocs différents. Le long de l'exécution de ces opérations, l'UC est mise en attente.

β) Soit pour générer une situation d'activité critique avec saturation du module.

γ) Soit pour générer une situation d'activité critique en saturant l'environnement du module.

δ) Soit pour générer une situation d'activité critique globale.

Dans l'exemple étudié en §III, on a distingué les facteurs qui sont de nature à saturer l'environnement d'un module canal et ceux qui sont de nature à saturer le canal même.

Nota : Un taux relativement faible d'erreurs dues à des situations de simultanéité et de charge peut être acceptable tant que ces erreurs sont détectables et récupérables. Le but de générer les diverses situations d'activité critique est précisément d'examiner le comportement de chaque module et de déterminer, compte tenu de la configuration, le taux de probabilité d'erreurs.

IV - 2. EXECUTION D'UNE TACHE DE TEST

On rappelle que chaque protest est composé de 3 sous-séquences. L'exécution d'une sous-séquence correspond à une *phase* de l'exécution du protest.

Dans le cas d'un protest d'E/S, la phase de vérification peut être scindée en deux sous-phases :

- Une sous-phase de vérification courte qui se réduit au traitement de l'acquittement de l'interruption de fin d'opération et au prélèvement du contenu des registres d'état d'E/S ;

- Une sous-phase de vérification longue qui analyse ces résultats afin d'en déduire un diagnostic.

Cas d'une tâche de test unitaire

Les protests de la tâche sont exécutés en séquence. Ces protests peuvent être ordonnés de façon à réduire au maximum les initialisations nécessaires pour chaque protest. A la fin de l'exécution de la tâche, un compte rendu ou "report de diagnostic" est fourni qui résume les résultats obtenus par l'exécution des protests.

Cas d'une tâche de test de simultanéité

Le test est axé sur l'exécution parallèle des opérations testées. Le parallélisme est dû au fait que pendant le déroulement d'une opération d'E/S, l'UC peut initialiser d'autres opérations d'E/S (pour d'autres protests) ou effectuer tout autre travail. Les phases qui représentent alors le plus d'intérêt dans l'exécution des protests d'E/S sont les phases d'exécution.

Par ailleurs, il est difficile de générer d'emblée une situation d'activité particulière de simultanéité au sein d'un module. Mais on peut approcher de telles situations en rebouclant un certain nombre n de fois sur les phases d'exécution des protests d'E/S de la tâche. Le nombre n peut être déterminé

de façon expérimentale ou choisi assez grand de manière à garantir, avec une forte probabilité, que la situation d'activité recherchée est atteinte au moins une fois.

L'exécution de la tâche s'effectue de la façon suivante :

a) Toutes les phases d'initialisation des protests d'E/S de la tâche sont exécutées en séquence, puis les phases d'exécution sont lancées simultanément (exécution des instructions de départ d'E/S pour les opérations correspondantes).

b) Pendant le temps d'exécution de ces opérations, l'UC peut être soit mise en attente, soit amenée à exécuter des protests UC choisis de façon à rendre critique la situation d'activité générée dans les modules sous test.

c) Quand une interruption d'E/S survient, les protests UC sont suspendus momentanément, la séquence de vérification courte est effectuée pour le protest d'E/S ayant été à l'origine de l'interruption et l'opération d'E/S qui vient de se terminer est de nouveau lancée. L'UC reprend ensuite son activité comme en b).

d) Quand toutes les phases d'exécution des protests d'E/S auront été effectuées au moins n fois, les phases de vérification longue sont effectuées pour tous les contextes de fin de phase prélevés lors des phases de vérification courte. Un compte rendu est fourni qui résume les résultats obtenus.

Le fait de différer les phases de vérification longue des protests d'E/S permet de maintenir, le plus longtemps possible, la situation de simultanéité recherchée par l'exécution de la tâche.

IV - 3. TRAITEMENT DES INCIDENTS

On peut distinguer deux types d'erreurs : les erreurs simples et les erreurs de simultanéité.

- Une erreur simple est une erreur qui est détectée lors de l'exécution d'un test unitaire. Les recoupements de protests permettent la déduction d'un certain diagnostic.

- Une erreur de simultanéité est une erreur qui ne se manifeste que lors d'une activité de simultanéité générée au sein des modules d'une sous-configuration donnée.

L'opération de "réduction" consiste, dans ce dernier cas, à cerner l'origine de l'erreur au niveau d'une sous-configuration minimale.

Algorithme de réduction

Supposons que la tâche de test de simultanéité ayant généré l'erreur renferme r protests : P_1, P_2,... P_r. Il s'agit de trouver les sous-tâches qui reproduisent la même erreur en ne mettant en jeu qu'un sous-ensemble minimal de ces protests.

1) On dira qu'une sous-tâche est *significative* pour une erreur donnée si son exécution reproduit cette erreur.

2) Une sous-tâche significative est *irréductible* si, en supprimant l'un quelconque des protests qu'elle renferme, son exécution ne reproduit plus l'erreur.

La réduction peut être effectuée par étapes successives :

- Chaque étape consiste à distinguer parmi les sous-taches renfermant p protests ($2 \leq p \leq r$) quelles sont les sous-taches à p-1 protests qui sont significatives;

- A la fin de cette étape, les sous-tâches irréductibles à p protests sont notées et seules les sous-tâches significatives à p-1 protests sont retenues pour l'étape suivante;

- La procédure s'arrête quand, à l'issue d'une étape, aucune réduction n'a pu être effectuée.

Des précautions sont à prendre pour tenir compte des cas d'erreurs intermittentes. Pour cela, une procédure de réduction ne doit être engagée que si la réitération de l'exécution de la tâche, pour un certain nombre N de fois, confirme la manifestation de la même erreur.

V - LE PROGRAMME SIMUX

V - 1. CRITERES TECHNIQUES DE REALISATION DU PROGRAMME

Modularité

Le choix d'une organisation modulaire du programme doit permettre de :

- Concentrer l'étude et la réalisation du programme séparément au niveau de sous-programmes (ou modules);

- Doter le programme d'une structure qui permet une mise au point aisée et rapide;

- Faciliter d'éventuelles modifications à apporter au cours de la réalisation du programme et après sa mise en service |D15|.

Les interfaces de certains de ces modules peuvent être normalisés pour permettre leur utilisation éventuelle par d'autres programmes de test pour des systèmes de la même gamme. Exemple : on peut prévoir un module de gestion de la console opérateur pour l'initialisation et le contrôle du programme SIMUX, qui peut être utilisé pour ces mêmes fins par d'autres programmes de test.

Portabilité

Ce critère est à interpréter dans le sens d'une adaptation du programme à des configurations diverses du système.

Automatisation

La reconnaissance de la configuration à tester, la génération des tâches de test pour une configuration donnée et la déduction d'un diagnostic en cas de détection d'erreur doivent être étudiées de manière à pouvoir être effectuées de la façon la plus automatisée possible. Ceci afin de minimiser toute nécessité d'une intervention humaine pour le déroulement du test.

Similitude avec une exploitation réelle

Il est intéressant que certaines activations de simultanéité en mode standard puissent simuler des conditions d'exécution considérées comme typiques lors d'une exploitation réelle. Cela permet de comparer les caractéristiques du comportement du système à celles d'autres systèmes existants et, éventuellement, d'en tirer des renseignements utiles concernant l'amélioration de nouveaux produits logiciels et matériels.

Facilités de test

Un certain nombre de facilités comme l'organisation des "traces", les points d'arrêt, les rebouclages sur les tâches de test,... doivent être prévues de manière à pouvoir adapter le programme aux circonstances spécifiques de chaque test (endurance, déverminage, effleurage..). Certaines de ces facilités sont détaillées dans la suite.

Reports de diagnostic

Les sorties de reports détaillés sur la console de maintenance doivent permettre la surveillance et le contrôle continus de l'évolution des processus de test.

V - 2. ARCHITECTURE DU PROGRAMME

La figure VI-5 illustre une architecture possible du programme. On distingue les modules suivants :

1) Le module GESCONS chargé de la gestion de la liaison avec la console opérateur.

2) Le module CONFIGURATEUR chargé de la reconnaissance automatique de la configuration opérationnelle du système soumis au test. Toutes les informations concernant les unités physiques composant la configuration (unités mémoire, unités de calcul, unités d'entrées/sorties...) ainsi que les connexions entre ces unités sont portées dans une "table de configuration" : TABCONF.

3) Le module ELABTEST qui utilise les informations préparées dans TABCONF pour élaborer de façon automatique les tâches de test à faire exécuter par le système. Il se sert pour cela d'un ensemble de séquences de code rangées dans une "table de segments de code de test" (voir § III).

Il est important que l'élaboration des tâches de test puisse tenir compte des résultats obtenus au fur et à mesure du déroulement des tests. Il est, en effet inutile, si l'exécution des premières tâches a permis de localiser une anomalie au niveau d'une unité du système, de continuer à utiliser cette unité pour d'autres tests. L'opérateur doit pouvoir décider s'il faut retirer cette unité de la table de configuration ou la maintenir.

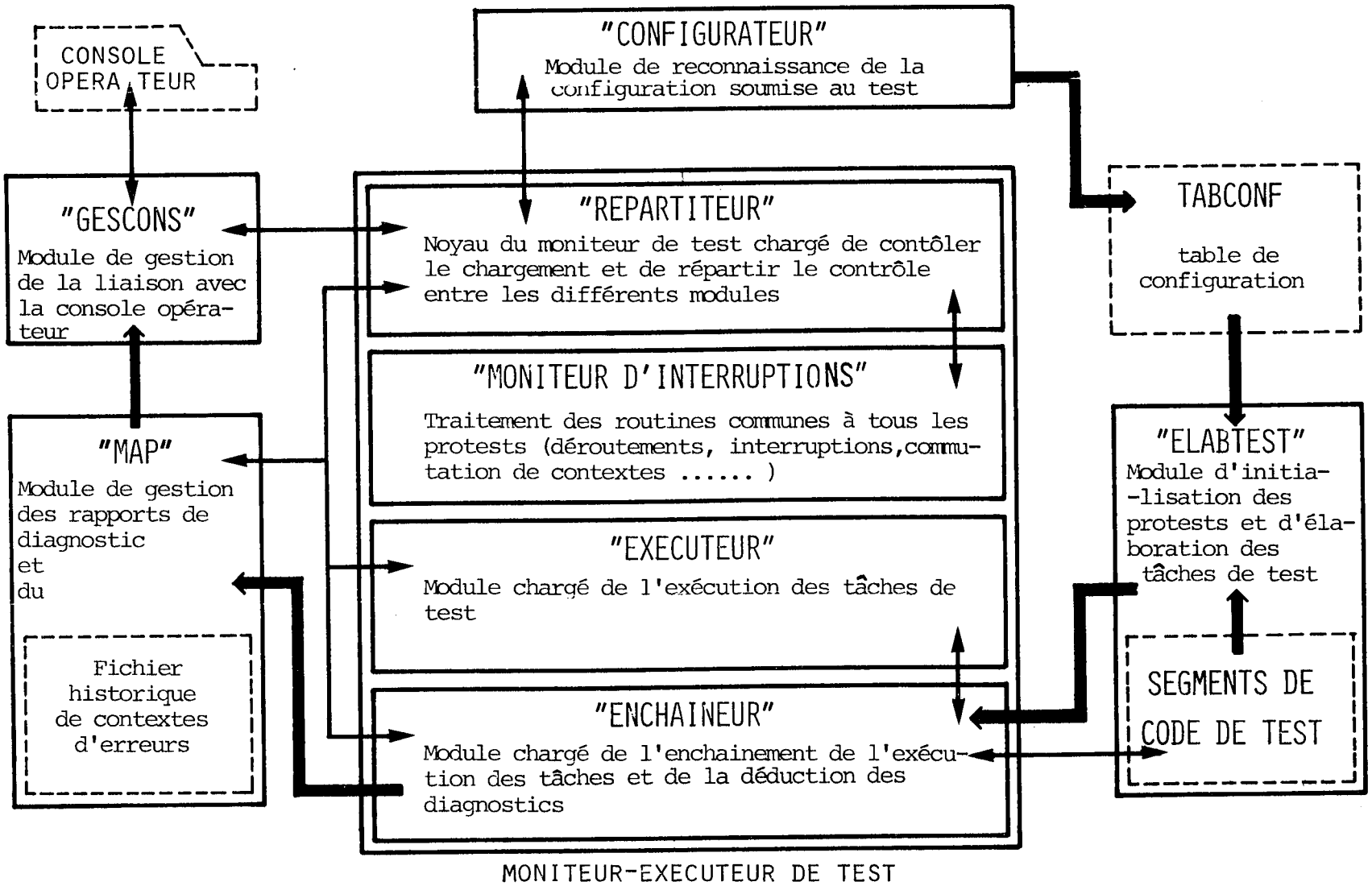


Figure VI-5

Organisation modulaire du programme

4) Le module MAP chargé de la gestion des messages de diagnostic et du "fichier historique de contextes d'erreurs". Il est quasiment impératif, afin de faciliter toute modification ultérieure du programme, que l'élaboration des messages et des reports de diagnostic soit centralisée dans un même module.

Le moniteur de test peut être découpé en 4 modules :

5) Le module REPARTITEUR qui constitue le noyau du programme chargé de contrôler l'opération de chargement initial (chargement du programme, initialisation du dialogue opérateur à travers le module GESCONS, initialisation du fichier historique dans le module MAP...), puis de superviser le passage du contrôle entre les différents modules.

6) Le MONITEUR D'INTERRUPTIONS qui regroupe les routines communes à tous les protests concernant le traitement des interruptions, le traitement des déroutements, la commutation des contextes machine et programme...

7) Le module EXECUTEUR chargé de gérer l'activation des protests pour chaque tâche de test, selon le type de test effectué par la tâche :

. Exécution séquentielle des protests dans le cas d'une tâche de test unitaire;

. Multiplexage de l'exécution des protests dans le cas d'une tâche de test de simultanéité.

8) Le module ENCHAINEUR dirige l'enchainement de l'exécution des tâches préparées par ELABTEST et analyse les résultats obtenus afin d'en déduire un diagnostic. Il se charge de conduire l'opération de réduction en cas de détection d'erreurs générées par l'exécution d'une tâche de test de simultanéité.

V - 3. MODES D'OPERATION

Le programme peut avoir trois modes d'opérations :

- Le mode d'opération "normal"
- Le mode d'opération "avec localisation"
- Le mode d'opération "en endurance".

L'opérateur doit pouvoir, à tout moment, demander le passage d'un mode d'opération à un autre.

- En mode normal, les tâches de test élaborées par ELABTEST sont exécutées une seule fois. Les erreurs qui peuvent être détectées sont comptabilisées dans un report de diagnostic (un report pour chaque tâche), mais aucune opération de localisation n'est engagée.

- Tant qu'aucune erreur n'est détectée, le mode d'opération avec localisation est identique au mode d'opération normal. Mais, en cas de détection d'erreur, une opération de localisation est engagée systématiquement afin de cerner au mieux son origine. (voir §IV-3, traitement des incidents).

- Le mode d'opération en endurance peut servir soit pour effectuer une action de déverminage en rebouclant un nombre assez grand de fois sur l'exécution de chaque tâche de test, soit pour prolonger une opération de localisation cherchant à cerner l'origine d'une erreur intermittente.

V - 4. REPORTS DE DIAGNOSTIC ET FACILITES DE TEST

V - 4.1. Reports de diagnostic

Les reports de diagnostic doivent fournir une information exhaustive et facile à interpréter. En mode d'opération normal ou avec localisation, un report doit être fourni à la fin de chaque exécution d'une tâche. En mode d'opération en endurance, un report global peut être fourni pour l'ensemble des itérations effectuées pour l'exécution d'une même tâche.

L'information fournie par un report doit comporter à chaque fois :

- Un numéro de report
- Une identification de la tâche exécutée
- La sous-configuration ayant pris part dans l'exécution de cette tâche
- Une indication du temps mis pour cette exécution
- Un bilan global des opérations UC et/ou d'E/S qui ont été effectuées
- Un rappel du mode d'opération en cours
- Un relevé des incidents parasites si occurence a eu lieu lors de l'exécution de la tâche : interruption provoquée par une unité ne faisant pas partie de la sous-configuration en test, passage d'une unité périphérique de l'état prêt à l'état non prêt (fin de papier sur imprimante, casier d'alimentation vide sur un perforateur de cartes...)

- Toute erreur ayant été détectée par l'exécution de la tâche est signalée en indiquant :

  le protest ayant généré l'erreur;

. la sous-configuration concernée par l'exécution de ce protest;
. le type d'erreur et les moyens qui ont permis sa détection (contrôle câblé, procédure logicielle, résultat obtenu différent du résultat attendu sur test d'état périphérique);

- un diagnostic final.

En mode d'opération en endurance, le report doit indiquer les conditions ayant mis fin au rebouclage sur l'exécution de la tâche (facteur de rebouclage atteint, dépassement d'un taux d'erreurs, déclanchement d'un compteur d'horloge, ect...).

En mode d'opération avec localisation, un report doit être fourni après chaque pas de réduction comportant, en plus des informations citées plus haut :

- Un numéro relatif du report depuis le déclenchement de l'opération de réduction
- La sous-configuration retenue à ce pas de l'opération de réduction
- Le résultat obtenu : réduction significative ou non significative
- Un diagnostic final si dernier pas de l'opération de réduction

V - 4.2. Facilité de trace

C'est ce qui donne la possibilité de reprendre l'exécution des tests à partir d'un point donné. Le numéro affecté à chaque report peut servir comme pointeur de point de reprise; le test reprend au stade où il était au moment de la sortie de ce report.

D'autres facilités peuvent être prévues concernant, par exemple, les possibilités de choix des tâches de test à faire exécuter avec enchaînement dirigé par l'opérateur, les possibilités de modification des paramètres des tâches (élimination de protests, modification des paramètres des protests : adresses mémoire utilisées, longueur de transferts effectués par des protests d'E/S, facteur de rebouclage...).

CHAPITRE VII

VALIDATION FONCTIONNELLE EN COURS DE CONCEPTION.

LE SYSTEME M.A.S.*1

*1. M.A.S. : Multilevel Assisted Design System.

Ce dernier chapitre sera consacré au problème de la validation fonctionnelle des systèmes en cours de conception. On rappellera brièvement la méthodologie de conception descendante généralement adoptée pour l'élaboration de nouveaux matériels et on mettra en évidence les caractéristiques des outils de modélisation et de simulation dont le concepteur a besoin pour effectuer les vérifications et évaluations nécessaires le long de sa démarche. On passera en revue certains des outils d'aide à la conception les plus utilisés à l'heure actuelle et on discutera de leurs possibilités et de leurs limites. On présentera, ensuite, le système MAS qui est un outil de modélisation et de simulation multiniveaux basé sur les réseaux de PETRI. On justifiera les choix adoptés pour la définition de cet outil et on montrera son originalité par rapport aux outils existants. Enfin, on donnera un exemple illustrant l'utilisation de cet outil à l'étude d'une structure multiprocesseur.

I - DEPUIS LA SPECIFICATION DU CAHIER DES CHARGES JUSQU'A LA REALISATION

Le cahier des charges est le document initial que l'on rédige pour spécifier ce que l'on attend d'un nouveau produit. Il est constitué essentiellement de deux parties :

- Les spécifications fonctionnelles qui décrivent l'ensemble des services que doit offrir le système. Cette description ne peut être dissociée de celle du comportement de l'environnement qui réagit sur le système,

- Les contraintes, qui expriment des desiderata d'ordre opérationnel et économique :

  . contraintes d'exploitation : performances max, min ; temps de réponse ; charge ; facilité d'emploi ; ...

  . contraintes de sécurité : prévention contre un fonctionnement "non sûr" que peut provoquer une altération de l'environnement ou une défectuosité du matériel constituant le système ; maintenabilité du système et possibilité de son fonctionnement dégradé ;

  . contraintes technologiques et de fiabilité ;

  . contraintes financières : coûts.

La tâche du concepteur est de concevoir le système qui satisfait à ces desiderata. La démarche adoptée peut être résumée dans les étapes suivantes [E7] :

A - Décomposition fonctionnelle et définition de l'architecture du système. Il s'agit de déterminer les sous-fonctions dont l'agencement judicieux permet de reconstituer les fonctions globales que doit réaliser le système. Ces sous-fonctions sont ensuite réparties en groupes de façon à implémenter les fonctions de chaque groupe dans un même module.

B - Etude des choix possibles de structure et des types de lien entre les modules de la structure . Ces choix sont faits en tenant compte des possibilités de parallélisme mises en évidence lors du découpage ainsi que de critères de sécurité (redondance, dédoublement de chemins fonctionnels, reconfiguration ...) et de performances (harmonie des caractéristiques dynamiques de chaque module avec celles des modules adjacents).

C - Etude de la conception interne de chaque module et précision des protocoles de synchronisation et d'échange avec les autres modules.

D - Elaboration des schémas logiques et choix des technologies de réalisation.

Cette démarche est "itérative" notamment pour les trois premières étapes. Le concepteur rompt avec la complexité du système en le définissant de façon progressive ; chaque pas apportant de nouvelles précisions à des niveaux de plus en plus fins. Certaines décisions prises à un niveau donné peuvent remettre en cause des décisions prises à des pas antérieurs ; il faut alors revenir en arrière pour modifier ces décisions, évaluer l'impact de ces modifications sur le comportement global du système et réajuster en conséquence les compromis entre les divers critères de sécurité, de performances et de coût. Cette démarche est aussi "récursive" dans le sens qu'elle est adoptée tant pour l'étude de la conception du système à partir de modules, que pour l'étude de la conception des modules à partir de sous-ensembles plus fins.

Pour mener à bien ce travail, le concepteur doit disposer d'un ensemble cohérent d'outils qui permettent de modéliser le système à plusieurs niveaux de détails et facilitent les vérifications et les évaluations effectuées à chacun de ces niveaux. Les caractéristiques de tels outils peuvent être résumées comme suit : [E1] [E2] [E3] [E4] [E5] [E6] [E7]

1) Permettre une description multiniveaux, assurant une transition cohérente entre les divers niveaux de la conception.

2) Coordonner les décisions prises le long de la conception. Le concepteur doit, notamment, pouvoir apprécier l'intêret des performances intrinsèques à chaque module, tenant compte des choix de structure adoptés, des types de lien entre les modules de cette structure et des protocoles d'échange sur ces liens, du degré de multiplexage, de la charge globale, ... Ces outils doivent par conséquent :

- . offrir les possibilités d'une description modulaire qui met en évidence la structure du système et facilite l'étude et l'analyse du comportement de chaque module ;
- . pouvoir représenter la coordination des évolutions parallèles synchrones et asynchrones entre modules et internes à chaque module ;
- . tenir compte du facteur temps pour l'évaluation des performances et la vérification de la cohérence dynamique de la structure étudiée.

3) De tels outils doivent aussi permettre un travail d'analyse visant à :

. détecter les situations de conflit dues au fonctionnement parallèle interne et entre modules ;
. mettre en évidence les situations de fonctionnement pouvant conduire à des blocages ou à des écroulements sensibles des performances ;
. étudier les solutions permettant de prévenir un fonctionnement incorrect dû à des effets parasites externes ou à la présence de pannes.

4) Enfin, il est tout aussi important que ces outils puissent disposer d'un nombre suffisant de concepts et de primitives qui permettent une description naturelle et directe du fonctionnement des systèmes.

II - REVUE CRITIQUE DES LANGAGES EXISTANTS D'AIDE A LA CONCEPTION DES SYSTEMES MATERIELS : CHDL*1

Une classification assez complète des langages CHDL existants, ainsi que des critiques intéressantes concernant leur utilisation peuvent être trouvées dans [E8], [E9] et [E10]. Le but ici est simplement de faire le point des possibilités offertes par les langages les plus utilisés à l'heure actuelle et de situer leurs limites afin de pouvoir les comparer par la suite à celles proposées par le système M.A.S.

II - 1. NIVEAUX DE DESCRIPTION

On peut distinguer trois niveaux principaux de description d'un système :

- Description de comportement : Le système est décrit comme une boite noire dont on connait seulement les relations entre ses variables d'entrée et ses variables de sortie. La façon dont ces relations sont décrites n'est pas nécessairement liée à une implémentation matérielle et l'aspect temporel est pris en compte de façon approximative, lorsqu'il n'est pas ignoré.

- Description fonctionnelle : Le système est décrit par la donnée de ses variables internes et de l'algorithme sur ces variables qui décrit son fonctionnement. Toutefois, les opérateurs utilisés peuvent ne pas correspondre à des primitives matérielles. L'aspect temporel est décrit à l'échelle des opérations qui

*1. CHDL : Computer Hardware Description Language.

composent l'algorithme, et le parallélisme et la synchronisation sont pris en considération.

- Description structurelle : Le système est décrit comme l'interconnexion de composants. L'algorithme de fonctionnement est donné de façon implicite en spécifiant le comportement de chaque composant. Etant très proche de la réalisation physique, le comportement temporel est exprimé en termes d'impulsions d'horloge et / ou d'évènements de synchronisation.

Au cours de la conception d'un système, les modules qui le composent passent par divers stades de définition qui correspondent à ces trois niveaux de description. La structure et le fonctionnement de chaque module sont précisés en parallèle et de façon progressive. L'outil qui semble alors le plus adéquat, afin d'assurer une transition cohérente entre ces divers niveaux, est celui qui permet de superposer une description structurelle à une description fonctionnelle multiniveaux. C'est à dire, quelque soit le niveau de description, la structure déjà établie en termes de modules et de liens entre modules est mise en évidence, et le fonctionnement de chaque module est décrit par la donnée explicite de l'ensemble de ses variables et de l'algorithme sur ces variables qui représente ce fonctionnement.

On constate, cependant, que les possibilités des trois niveaux de description apparaissent comme contradictoires dans les langages CHDL existants. La plupart de ces langages tendent à être spécifiques à une description structurelle fine (niveau transfert de registres ou circuits CASSANDRE [E11], DDL [E12], CDL [E13]). A l'autre extrémité, des langages moins spécialisés pour la description du matériel tels que GPSS [E14], SIMULA [E15], SIMSCRIPT [E16] sont utilisés pour l'évaluation des performances au niveau d'une description de comportement [E17] [E22]. Le pont entre ces deux niveaux de description est resté difficile à faire en l'absence de langages qui s'adaptent convenablement à une description fonctionnelle progressive [E18] [E19] .

II - 2.DESCRIPTION DU PARALLELISME ET DE LA SYNCHRONISATION

Une des caractéristiques importantes d'un langage CHDL est sa faculté de décrire le parallélisme et la synchronisation inhérents au fonctionnement du matériel. Ceci est généralement fait de la façon suivante : chacune des actions qui composent les fonctions du système est décrite à l'aide d'une séquence d'instructions et d'une "variable de contrôle" (étiquette) qui pointe

cette séquence et indique les conditions de son exécution. Ces conditions portent sur les variables du système et sont testées en permanence le long de la simulation ; chaque fois qu'elles sont vérifiées, les instructions associées sont exécutées.

Cette façon de faire procède du concept de la séparation des partie contrôle et partie opérative d'un système (ou d'un algorithme). Tout système peut être décomposé en deux parties (sous-systèmes) connectées en tourbillon (figure VII-1) [E20] [E24]

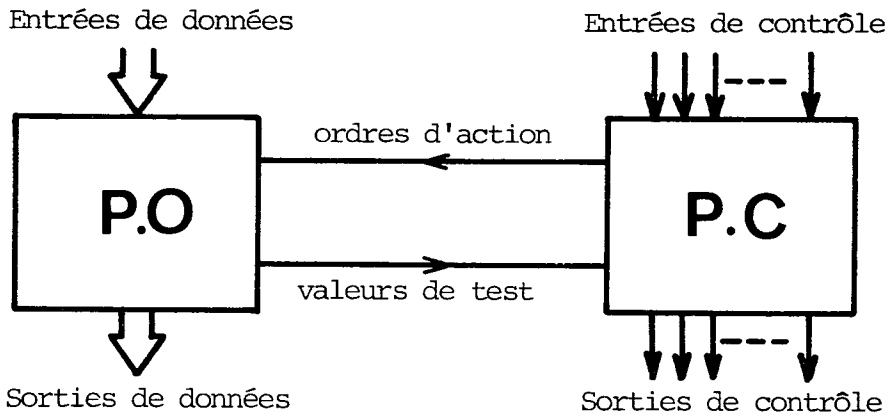


Figure VII-1

- La Partie Opérative (P.O) englobe l'ensemble des registres de données (variables) et les opérateurs qui sont utilisés soit pour calculer des valeurs de test sur ces données soit pour les transformer.

- La Partie Contrôle (P.C) gère le séquencement des opérations dans la P.O. Elle peut être représentée par un automate de Moore tel que à chaque état est associé un ensemble d'actions exécutables simultanément (compatibles) dans la P.O. Le fonctionnement du système est tel que le positionnement de la P.C à un état est interprété par la P.O comme l'ordre d'exécution des actions associées à cet état (ordre d'activation d'opérateurs). La P.C. évolue d'un état s à un état s' en fonction des changements d'état des entrées de contrôle externes et des valeurs de test calculées dans la P.O.

Ce fonctionnement est décrit dans le programme en associant à chaque paquet d'instructions décrivant une action donnée du système, une variable de contrôle qui exprime la condition d'activation de cette action en fonction de l'état de la P.C et des valeurs de ses entrées. Les langages qui permettent directement ce genre de programmation sont dits du type "non-procéduriel" [E8] par opposition à "procéduriel" pour les langages dans lesquels les instructions sont exécutées dans l'ordre d'écriture, sauf branchement explicite.

Il se trouve que les langages non-procéduriels existants se situent au niveau d'une description structurelle fine, où les actions décrites correspondent à des primitives matérielles élémentaires de type booléen ou affectation de registre. (CASSANDRE, DDL, CDL). Ceci rend leur utilisation peu envisageable pour la description de systèmes de taille importante. Aussi, si cette distinction P.C - P.O est observée pour l'analyse et la description du système, elle apparait beaucoup moins une fois le programme écrit, puisque les deux structures de contrôle et de traitement de données sont imbriquées. Pour des systèmes complexes, et notamment quand il s'agit de décrire le parallélisme à plusieurs niveaux, les programmes deviennent peu lisibles et se prêtent difficilement à la modification et à la validation. [E23] [E24] [E25]

Les concepts de la séparation P.C - P.O et de l'interprétation non-procédurielle du fonctionnement du système peuvent être mieux exploités dans le cadre d'une description fonctionnelle multiniveaux :

- La connaissance explicite de l'algorithme de fonctionnement du système, quelque soit le niveau de détail de la description, permet de maintenir une séparation nette entre les parties P.C et P.O. Il devient alors possible de suivre les évolutions de la P.C en ne s'intéressant qu'aux valeurs de certaines variables de la P.O à des instants précis : Au moment du positionnement de la P.C à un état donné, on sait quelles sont les variables sur lesquelles vont porter les transformations effectuées dans la P.O. On sait également sur quelles variables portent les prédicats des transitions à partir de cet état (exploitation de la notion de "réceptivité").

- Selon le niveau où l'on se place, on peut associer à chaque état non plus une action simple mais une procédure, tant que l'évolution de la P.C ne dépend pas des valeurs intermédiaires des variables le long de l'exécution de cette procédure. Ceci permet une description multiniveaux en explicitant chaque fois dans la P.O, et respectivement dans la P.C, uniquement les détails significatifs pour le niveau de description recherché.

- Enfin on peut limiter l'interprétation non procédurielle à la partie du programme qui décrit séparément la P.C, en donnant la possibilité de décrire les transformations de données dans la P.O à l'aide d'un langage procéduriel.

III - LE SYSTEME M.A.S.

III - 1. PRESENTATION

M.A.S (Multilevel Assisted Design System) est un outil de validation fonctionnelle multiniveaux basé sur les Réseaux de PETRI (RdP) [E26]. Sa définition a été menée de façon à satisfaire au mieux les besoins en outils de modélisation et de simulation formulés en I, tout en évitant les écueils constatés dans les outils existants.

Un système est décrit comme l'interconnexion d'un ensemble de modules qui coopèrent. La séparation nette des partie contrôle et partie opérative est observée dans la description de chaque module. Les actions dans la P.O sont décrites à l'aide d'un langage procéduriel, en l'occurence APL [E36]. Le choix d'APL se justifie par la nécessité de disposer d'un ensemble de primitives permettant aussi bien une description du niveau transfert de registres qu'une description algorithmique condensée. La P.C est décrite à l'aide d'un sous-langage du type non-procéduriel qui permet de traduire directement une représentation de l'automate de contrôle sous forme de RdP. L'utilisation d'un tel modèle facilite la description du parallèlisme et de la synchronisation et offre des possibilités intéressantes d'analyse [E27] [E30] [E31] [E32] [E34].

III - 2. UTILISATION DES RESEAUX DE PETRI POUR LA MODELISATION DES SYSTEMES

Dans ce paragraphe, on rappellera la définition des réseaux de PETRI, et on introduira certaines extensions qui permettent de mieux adapter l'utilisation de cet outil à la modélisation des systèmes (aspect temporel, synchronisation par des évènements externes, interprétation).

III - 2.1. Réseau de PETRI : RdP [E27]

Définition 1 :

Un RdP est un quadruplet $Q = (P, T, R, M_o)$ où

$P = \{P_1, P_2, \ldots, P_m\}$ est un ensemble fini de places ($P \neq \phi$) ;

$T = \{t_1, t_2, \ldots, t_n\}$ est un ensemble fini de transitions ($T \neq \phi$ et $P \wedge T = \phi$) ;

R = sous-ensemble de $\{P \times T\} \cap \{T \cap P\}$;

et $M_o : P \rightarrow N$, une application de P dans l'ensemble des entiers non-négatifs, appelée marquage initial.

Représentation :

On représente un RdP par un graphe orienté dont les noeuds sont les éléments de $P \cup T$. Une place est représentée par un cercle et une transition est représentée par un trait (figure VII.2). Il y a un arc orienté de P_i à t_j (respectivement de t_j à p_k) ssi $(p_i, t_j) \in R$).

On définit, de plus, des objets appelés marques pouvant se déplacer dans le réseau. On représente une marque par un point et on représente le marquage initial M_o en mettant dans chaque place p, $p \in P$, $M_o(p)$ marques.

exemple :

$P = \{p_1, p_2, \ldots, p_7\}$;

$T = \{t_1, t_2, \ldots, t_5\}$;

$R = \{(p_1,t_1), (p_2,t_2), (p_3,t_2), (p_4,t_3), (p_5,t_4), (p_6,t_4), (p_7,t_5)\} \cup \{(t_1,p_2), (t_2,p_1), (t_2,p_4), (t_3,p_5), (t_3,p_6), (t_4,p_7), (t_5,p_3)\}$

$M_o(p_1) = M_o(p_3) = 1$; $M_o(p_i)_{i=2,4,5,6,7} = 0$

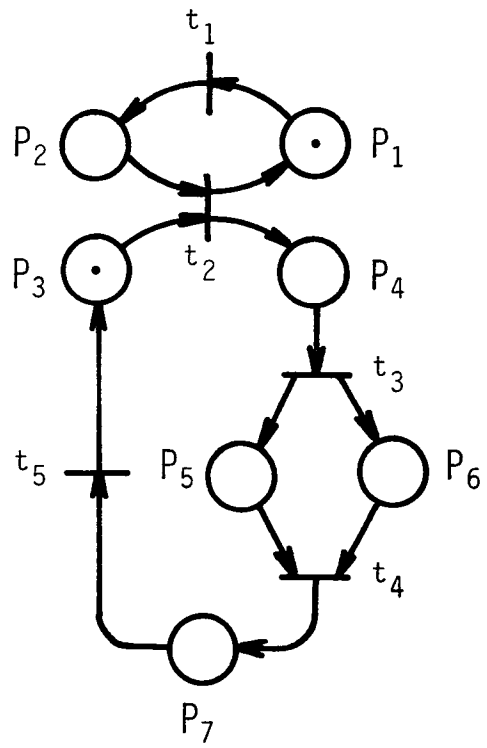


Figure VII-2

Dans la suite, on notera par ${}^{\bullet}t_j$ l'ensemble des places d'entrée (prédécesseurs immédiats) de t_j, et par $t_j^{\bullet}$ l'ensemble des places de sortie (successeurs immédiats) de t_j. Dans l'exemple donné, ${}^{\bullet}t_2 = \{p_2, p_3\}$ et $t_2^{\bullet} = \{p_1, p_4\}$.

règles d'évolution des marques :

On dira qu'une place est pleine si elle contient au moins une marque et vide sinon. Une transition est validée ssi toutes ses places d'entrée sont pleines.

- Toute transition validée peut être mise à feu. Cette mise à feu consiste à enlever une marque de toute place d'entrée de la transition et à mettre une marque dans toute place de sortie.

- Quand une mise à feu d'une transition t_j commence, une marque est réservée dans chaque place d'entrée et devient transparente pour toute autre transition. A la fin de la mise à feu, ces marques sont retirées des places d'entrée et une marque est mise dans chaque place de sortie.

Définition 2 :

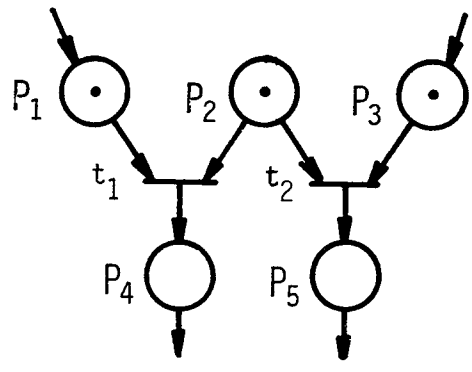
On dira qu'un RdP est sauf si : $\forall\ p \in P$, on a $M_o(p) \leq 1$ et si quelle que soit la séquence de mises à feu effectuée depuis le marquage initial, on n'a jamais la situation où une place contient plus qu'une marque.

Définition 3 :

On dira qu'un RdP sauf est sans conflit si, en aucun cas, on ne peut avoir la situation où deux transitions distinctes t_i et t_j telles que ${}^{\bullet}t_i \cap {}^{\bullet}t_j \neq \phi$ sont susceptibles d'être mises à feu simultanément.

Un exemple de situation de conflit dans un RdP sauf est le suivant :

${}^{\bullet}t_1 \cap {}^{\bullet}t_2 = \{p_2\}$



Définition 4 :

On dira qu'un RdP est vivant si :

. ∀ la séquence de mises à feu préalablement effectuée depuis le marquage initial, et

. ∀ $t_j \in T$, on peut toujours organiser une séquence de mises à feu qui rend susceptible la mise à feu de t_j.

Les définitions 2,3 et 4 sont données de façon à pouvoir être généralisées pour tous les RdP particuliers que l'on va définir.

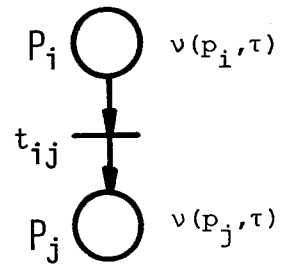
III - 2.2. Réseau de PETRI Temporisé : RdPT

Définition 5 :

Un RdPT est défini par un sextuplet (P, T, R, M_o, T, ν) où :

- (P, T, R, M_o) est un RdP ;
- T est un sous-ensemble complètement ordonné de $\mathbb{R}$, appelé base de temps ; ($\mathbb{R}$ = ensemble des réels)
- $\nu : P \times T \to T$ telle que si $\nu(p, \tau_i) = \tau_j$ alors $\tau_j \geq \tau_i$.

On représente un RdPT par le RdP associé en indiquant sur chaque place p l'application $\nu(p, \tau)$.



règles d'évolution :

Une marque dans un RdPT peut se trouver dans l'un des deux états disponible ou indisponible. Initialement, toute place p contient $M_o(p)$ marques disponibles

- Une transition t_j est validée si toutes ses places d'entrée contiennent au moins une marque disponible. Toute transition validée peut être mise à feu.

- La mise à feu d'une transition validée t_j consiste à enlever une mar-

que disponible de chaque place d'entrée et à mettre une marque indisponible dans chaque place de sortie.

- une marque reste indisponible dans une place p durant l'intervalle de temps entre l'instant τ_0 de son arrivée à la place et l'instant $\nu(p,\tau_0)$, puis devient disponible.

Remarque :

Pour une place p, la fonction $\nu(p,\tau)$ n'est définie que pour les valeurs $\tau_j \in T$. Il se pose alors le problème de la durée de mise à feu des transitions ; toute mise à feu doit avoir unedurée cadrée sur un intervalle $[\tau, \tau_j]$ tel que $\tau_j \in T$. Le problème se résoud de lui même si $T = \mathbb{R}$ (donc ν partout définie). Si $T \neq \mathbb{R}$, on peut convenir de considérer que toute mise à feu est instantanée et qu'elle doit être effectuée dès la validation de la transition.

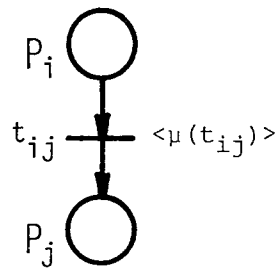
III - 2.3. Réseau de PETRI Synchronisé : RdPS

Définition 6 :

Un RdPS est défini par un sextuplet (P, T, R, M_0, E, μ) où

- (P, T, R, M_0) est un RdP ;
- $E = \{e_1, e_2, \ldots, e_s\}$ est un ensemble d'évènements externes ;
- $\mu : T \to E$.

On représente un RdPS par le RdP associé en portant sur chaque transition t_j l'étiquette $<\mu(t_j)>$ où $\mu(t_j)$ est l'évènement associé à t_j.



règles d'évolution :

Les règles d'évolution sont les mêmes que dans un RdP à la différence que la mise à feu d'une transition validée t_j ne peut avoir lieu qu'au moment de l'occurence de $\mu(t_j)$. Aussi, cette mise à feu est instantanée.

III - 2.4. Réseau de PETRI Temporisé Synchronisé : RdPTS

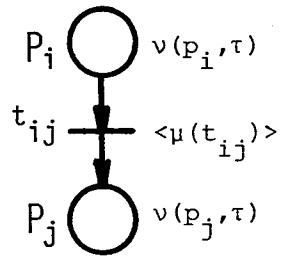
Définition 7 :

Un RdPTS est la superposition

- d'un RdPT (P, T, R, M_o, τ, ν)

et - d'un RdPS (P, T, R, M_o, E, μ), construits à partir d'un même RdP (P, T, R, M_o). De plus, les évènements de E sont tels que leur occurrence ne peut avoir lieu qu'à des instants $\tau_j \in T$.

On représente un RdPTS par le RdP associé en indiquant sur chaque place p l'application ν (p, τ) et sur chaque transition t_j l'évènement μ (t_j).



règles d'évolution :

Les règles d'évolution sont obtenues en superposant celles définies pour un RdPT et un RdPS.

- Une marque dans une place p peut se trouver dans l'un des deux états disponible ou indisponible. Initialement, toute place p contient M_o (p) marques disponibles.

- Une transition t_j est validée si toutes ses places d'entrée contiennent au moins une marque disponible. Mais la mise à feu d'une transition validée ne peut avoir lieu qu'au moment de l'occurence de l'évènement associé $\mu(t_j)$.

- La mise à feu d'une transition t_j est instantanée. Elle consiste à enlever de chaque place d'entrée une marque disponible et à mettre dans chaque place de sortie une marque indisponible.

- Une marque reste indisponible dans une place p durant l'intervalle de temps entre l'instant τ_o de son arrivée à la place et l'instant $\nu(p, \tau_o)$,

puis devient disponible.

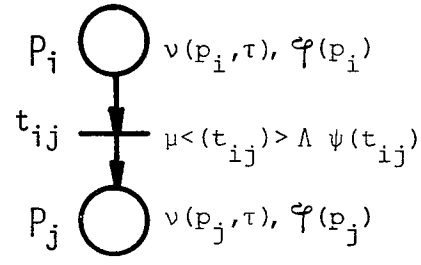
III - 2.5. Réseau de PETRI Interprété : RdPI

Définition 8 :

Un RdPI est défini par la donnée

- d'un sous système opératif (D, OP, C) tel que :
  - $D = \{d_1, d_2, \ldots, d_u\}$ est un ensemble fini de variables de données prenant leurs valeurs respectivement dans des espaces $D_1, D_2, \ldots, D_u$;
  - $OP = \{op_1, op_2, \ldots, op_v\}$ est un ensemble fini d'opérateurs qui effectuent des transformations sur les données, ces opérateurs sont définis comme des applications internes dans D ;
  - $C = \{c_1, c_2, \ldots, c_r\}$ est un ensemble de conditions (prédicats) sur les valeurs des données d_v.
- d'un RdPTS $(P, T, R, M_o, T, \nu, E, \mu)$;
- d'une application $\varphi : P \to OP$;
- et - d'une application $\psi : T \to C$.

On représentera un RdPI par le RdPTS associé en indiquant sur chaque place p_i l'opérateur $\varphi(p_i)$ et sur chaque transition t_j la condition $\psi(t_j)$. (On ne s'intéresse pas à associer une représentation précise au sous-système opératif).



règles d'évolution :

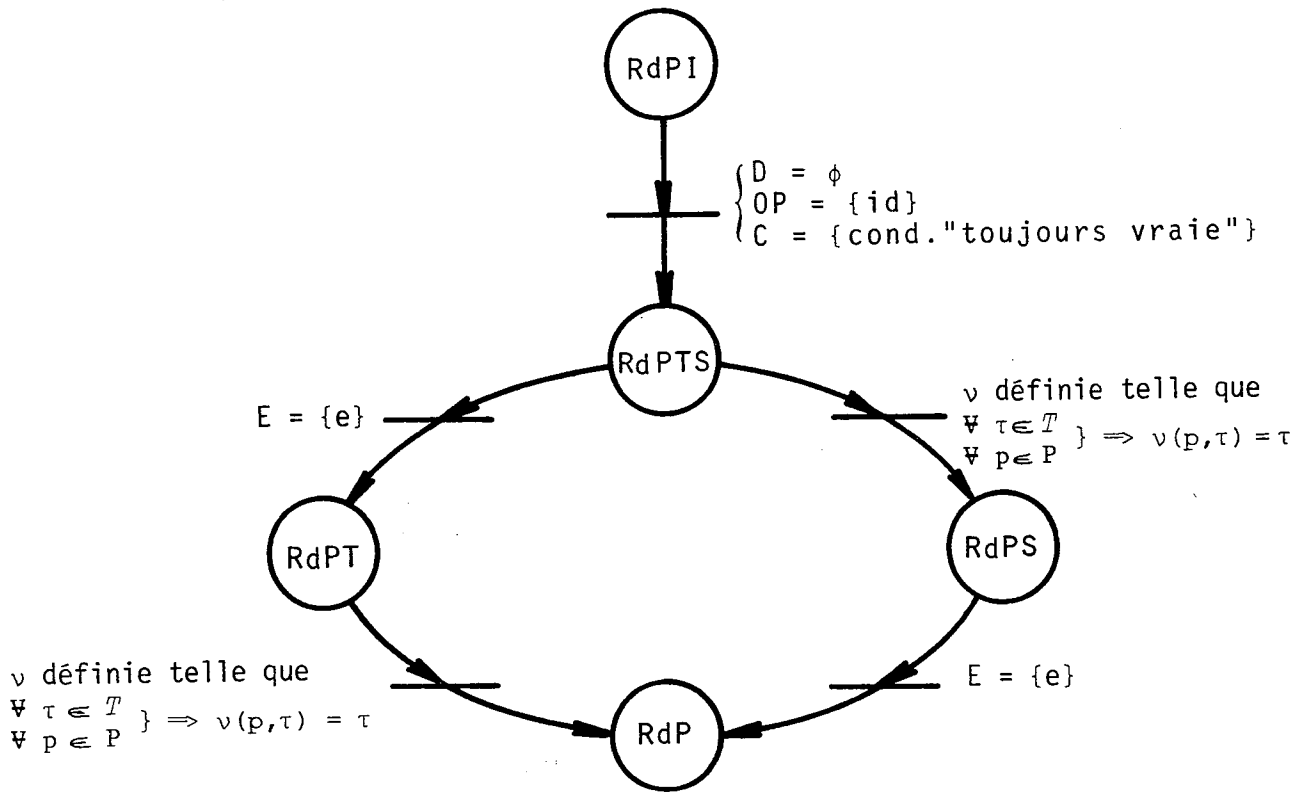
L'interprétation définie par le sous-système opératif (D, OP, C) et les applications φ et ψ introduit deux précisions supplémentaires aux règles d'évolution établies pour un RdPTS :

- La mise à feu d'une transition validée t_j n'est autorisée que si la condition ψ (t_j) est vérifiée. En tout cas, cette mise à feu ne peut avoir lieu qu'au moment de l'occurence de l'évènement μ (t_j).

- L'arrivée d'une marque dans une place p active l'opérateur $\varphi(p)$. Cette marque reste indisponible durant l'intervalle de temps entre l'instant τ_0 de son arrivée à la place et l'instant ν (p, τ_0), puis devient disponible.

III - 2.6. Conclusion - Utilisation des RdP pour la description des systèmes logiques.

Partant de la définition initiale des RdP, on a introduit certaines particularités qui ont permisde définir d'une part les RdPT et d'autre part les RdPS, puis progressivement les RdPTS et RDPI. La figure ci-dessous illustre les relations entre ces divers types de RdP : Un RdPTS tel que l'ensemble E des évènements externes est réduit à l'évènement trivial e ("toujours présent") est un RdPT. Un RdPTS défini avec ν telle que : $\forall\ \tau \in T$ et $\forall\ p \in P$; $\nu\ (p, \tau) = \tau$ (une marque devient disponible dès son arrivée à une place) est un RdPS, ...



légende
e = évènement trivial (toujours présent)
id = opérateur identité

Figure VII-3

Tout système peut être décrit par un RdPI. En effet, on peut toujours décrire la partie opérative d'un système à l'aide du sous-système opératif {D, OP, C} tel que :

D = ensemble des registres de données du système ;

OP = ensemble des opérateurs du système, auquel on adjoint l'opérateur id ; (voir note 1)

et C = ensemble des tests effectués sur le contenu des registres du système, auquel on adjoint le test "toujours vrai" - (voir note 2)

Le système global peut être décrit par un RdPI construit à partir de ce sous-système opératif, d'un RdPTS $(P, T, R, M_o, T = \mathbb{R}, \nu, E, \mu)$ sauf et des applications $\varphi : P \to OP$ et $\psi : T \to C$ définis de la façon suivante :

- L'ensemble E des évènements externes est défini par l'ensemble des changements d'état possibles des entrées de contrôle externe, auquel on adjoint l'évènement e ("toujours présent") - (voir note 3)

- (P, T, R, M_o) est le RdP qui décrit le schéma de contrôle de l'algorithme de fonctionnement du système. Cette description est donnée en associant à chaque place p un opérateur $\in$ OP, et en associant à chaque transition, d'une part, la condition $\in C$ (ou le test) qui autorise la mise à feu de cette transition, d'autre part, l'évènement $\in E$ qui force cette mise à feu. Les opérations $\varphi : P \to OP$, $\psi : T \to C$ et $\mu : T \to E$ se trouvent ainsi implicitement définies.

- L'arrivée d'une marque dans une place p déclenche l'éxécution de l'opération associée dans la P.O. La fonction ν (p, τ) est définie de façon à permettre de déterminer la durée de cette exécution.

Enfin, pour que le RdPI ainsi construit puisse représenter correctement le fonctionnement du système, on impose la contrainte supplémentaire que toute mise à feu d'une transition t_j doit avoir lieu dès la première occurence de l'évènement μ (t_j) depuis que cette mise à feu a été autorisée.

note 1 : *L'opérateur id sera associé à toute place de synchronisation (dans le RdPTS représentant la P.C du système) pour laquelle il ne correspond aucune action dans la P.O.*

note 2 : *La condition "toujours vraie" sera associée à toute transition dont la mise à feu, quand elle est validée, n'est assujettie qu'à*

l'occurrence d'un évènement externe.

note 3 : *L'évènement e sera associé à toute transition dont la mise à feu est effectuée dès l'instant où elle est validée et que la condition associée est vérifiée (indépendamment de l'occurrence de tout évènement externe).*

L'approche de description des systèmes logiques à l'aide des RdPI est intéressante à plusieurs points de vue :

- La grande facilité qu' apportent les RdP à la description du parallélisme et de la synchronisation, associée à une représentation simple et concise qui permet de visualiser l'évolution du contrôle dans le fonctionnement des systèmes [E27] [E30] [E35]

- Les RdP sont fondés sur une théorie mathématique dont certains développements ont été déjà appliqués à l'étude des problèmes de conflit et de blocage pour des systèmes fonctionnant en parallèle et partageant des ressources communes [E29] [E32] [E33] [E35]

- Le côté rigoureux d'une telle représentation en fait un remarquable outil d'analyse qui, de plus, permet de conserver tout le long d'une étude de conception le même support de raisonnement, chose qui n'était pas possible avec les outils d'aide à la conception existants [E31] [E32] [E34]

- On montrera par la suite, en présentant le langage MAS, comment la simulation des RdPI permet l'étude du comportement dynamique de systèmescomplexes et l'évaluation de leurs performances.

III - 3. LE LANGAGE M.A.S.

Comme on l'a déjà annoncé en III-1, un système est décrit en MAS comme l'interconnexion de modules fonctionnels. Chaque module étant représenté par un RdPI, on associe à chaque place p une procédure décrite en APL qui représente la transformation effectuée par l'opérateur $\varphi(p)$; la description de la P.C qui gère l'activation de ces procédures est donnée de façon séparée.

La prise en compte de façon propre de la synchronisation de la P.C à l'aide d'évènements externes compliquerait vainement l'interprétation du programme pour le niveau de description envisagé. Dans la version définie du langage

[E38] [E39], on considère que toute les entrées du système, (ou d'un module), qu'elles soient de contrôle ou de données, sont mémorisées dans des variables de la P.O et testées comme des conditions sur les transitions de la P.C.

III - 3.1. Structure générale d'un programme

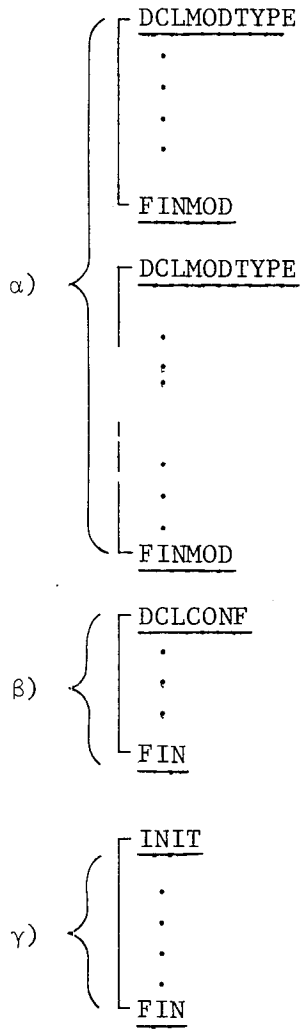
Un programme de description d'un système se compose de trois parties:

α) Définition des modules-types constituant le système.

β) Description de l'interconnexion de tels modules définissant la configuration du système.

γ) Initialisation du système, et spécification des séquences d'entrées pour lesquelles on veut étudier son comportement.

On donnera des précisions sur chacune de ces trois parties délimitées comme suit :



III - 3.2. Description d'un module type

La schéma ci-dessous montre un exemple de déclaration d'un module-type. Cette déclaration a comme paramètres les variables d'interface du module et contient :

. la déclaration des variables internes du module
. la description de la partie opérative (ensemble de procédures)
. la description de la partie contrôle

```
DCLMODTYPE  MODELE (VINT1, VINT2, VINT3 [ ], ... )
    DCL   VAR1, VAR2, VAR3 ;
          TAB [15 ; 2] ← 0 ;                    } déclaration des variables internes
            .
            .
            .
    DCLPROC  P1                                 }
         TEMP 5                                 }
         VINT1 ← 3                              }
            .                                   }
            .                                   }
            .                                   }
    FIN                                         }
    DCLPROC P2                                  }
         TEMP  ?9                               }
         VINT2 ← 5                              } Partie - Opérative
            .                                   }
            .                                   }
            .                                   }
    FIN                                         }
    DCLPROC P3                                  }
         TEMP  (VAR1-VAR2)/2                    }
            .                                   }
            .                                   }
            .                                   }
         T  ←  TIME                             }
    FIN                                         }
    DCLCTRL                                     }
      P_i1,P_i2, ...  ,P_in  SI  C_ij  ALORS  P_j1,P_j2, ....,P_jk } Partie - Contrôle
            .
            .
FINMOD
```

Les variables d'interface, comme les variables internes, peuvent être simples ou indicées.

a) Partie Opérative

Le nom de la procédure placé après le mot-clé DCLPROC est le nom de la place dans la partie contrôle à laquelle cette procédure est associée. Les instructions utilisées dans la description du corps de la procédure sont un sous-ensemble des instructions de APL [E37] enrichi d'une primitive de temporisation TEMP dont le paramètre indique le temps de simulation associé à l'exécution de la procédure. Ce paramètre peut être une constante ou une expression des variables de la procédure.

Dans l'exemple donné, l'assignation de VINT1 à la valeur 3 dans P1 est effectuée 5 unités de temps après l'activation de cette procédure. Dans P2, le temps d'éxécution est aléatoire,compris entre 1 et 9.

La primitive TIME à l'intérieur d'une procédure donne le temps courant au moment de l'activation de la procédure.

b) Partie Contrôle

Le mot-clé DCLCTRL annonce le début de la description de la partie contrôle. L'instruction de base utilisée a le format suivant :

$$P_{i1}, P_{i2}, \ldots \quad P_{in} \quad \underline{SI} \quad C_{ij} \quad \underline{ALORS} \quad P_{j1}, P_{j2}, \ldots \quad P_{jk}$$

Elle décrit une transition portant la condition C_{ij} et ayant pour places d'entrée P_{i1}, P_{i2}, ..., P_{in} et pour places de sortie P_{j1}, P_{j2}, ..., P_{jk}

(figure VII. 4)

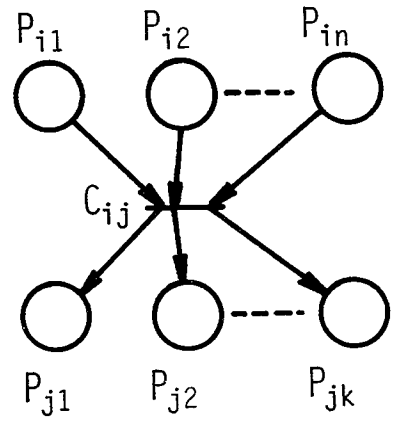


Fig.VII-4

- La description de la situation illustrée en figure VII.5 est donnée sous la forme :

1 SI C ALORS P_i

Dans ce cas, la condition C est testée en permanence, et chaque fois qu'elle est vérifiée, une marque indisponible est placée dans P_i

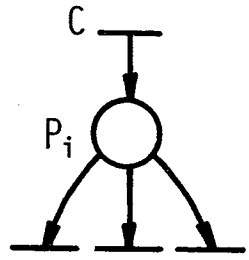


Fig.VII-5

- La situation illustrée en figure VII.6 est décrite de la façon suivante :

P_i SI C ALORS P'_i

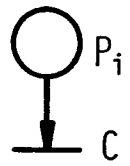


Fig.VII-6

- La syntaxe du langage permet la description, dans une même instruction, de plusieurs transitions ayant une place d'entrée unique.
Exemple : La situation illustrée en figure VII.7 peut être décrite par :

P_i SI C_{j1} ; C_{j2} ; ... ALORS P_{j1} ; P_{j2} ; .....

Lors de l'interprétation de cette instruction, une priorité est donnée à la première transition (dans l'ordre d'écriture de gauche à droite) dont la condition est vérifiée.

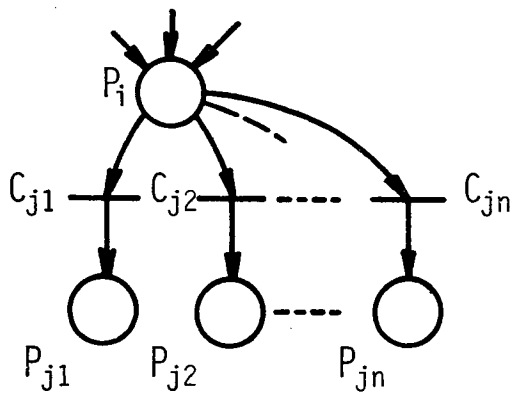


Fig.VII-7

- On peut associer à plusieurs places une même procédure paramétrée. Pour la même situation illustrée en figure VII.7, on peut écrire :

P_i SI C_{j1} ; C_{j2} ; ... ALORS P(J)

Lors de l'interprétation, le paramètre J prend la valeur entière correspondant au rang (en comptant à partir de 1) de la première condition vérifiée.

- Enfin, on donne dans MAS la possibilité d'associer des prédicats temporels aux transitions. Ainsi l'instruction :

P_i SI $!t_c$ ALORS P_j

signifie que la mise à feu de la transition portant le prédicat $!t_c$ n'est effectuée que ssi cette transition reste validée pendant un temps égal à t_c (figure VII.8).

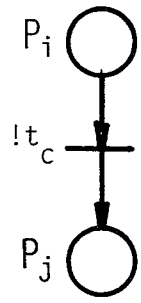


Fig.VII-8

III - 3.3 Description de la configuration d'un système :

Cette section du programme est délimitée par les mots-clé DCLCONF et FIN, et est constituée de deux sous-sections :

```
DCLCONF
    CREE
    MODULE1, MODULE2 := MODELE ;
      :
    LIE
    MODULE1 (VINT11, VINT21, ...) → MODULE2 (VINT12, VINT22, ...) ;
    MODULE1 (VINT31, ...) <—> MODULE2 (VINT32, ...) ;
      :
FIN
```

Dans la première sous-section, on crée les modules de la configuration du système à partir des modules-types prédéclarés. Dans l'exemple donné, deux modules MODULE1 et MODULE2 sont crées à partir du module-type MODELE.

Dans la deuxième sous-section, on établit les liens entre les variables d'interface de ces modules. Ces liens peuvent être de deux types : unidirectionnels (→) ou bidirectionnels (<—>).Dans l'exemple, les changements de valeur des

variables VINT11, VINT21, ... du module MODULE1 affectent directement, respectivement, les variables VINT12 et VINT22, ... du module MODULE2, mais la situation inverse n'est pas vraie. Par contre, les variables VINT31, ... du module MODULE1 et VINT32, ... du module MODULE2 sont considérées, respectivement,comme confondues.

III - 3.4. Définition de l'état initial du système et des valeurs d'entrée pour la simulation.

Cette dernière section du programme est composée de deux sous-sections annoncées par les mots-clé INIT et DONNEES

```
INIT
   MODULE1 (VAR1, VAR2, ...) = (valeur1,  valeur2, ....) ;
   MODULE1.VAR3   = Valeur3 ;
     :
DONNEES
     Tj   MODULE1( VAR1) = Valeur1,  MODULE2( VAR2) = Valeur2,  ... ;
             :
     Tk  + Tp    MODULE1( VAR3) = Valeur3 ;

     Ts  STOP ;
     :
FIN
```

La première sous-section définit l'état initial du système. Dans cet exemple, MODULE1(VAR1, VAR2, ...) = (Valeur1, Valeur2,) signifie que les variables VAR1, VAR2, ... du module MODULE1 sont initialisées respectivement aux valeurs Valeur1, Valeur2, ... L'initialisation peut porter soit sur les variables de données de la P.O, soit sur les variables représentant les places dans la P.C. Toutes les variables non initialisées prennent par défaut la valeur zéro.

La deuxième partie donne la séquence des entrées pour laquelle le comportement du système est simulé. Chaque élément de la séquence est présenté sous le format suivant :

Tj MODULE1(VAR1) = Valeur1, MODULE2(VAR2) = Valeur2, ... ;

Ceci signifie qu'à l'instant Tj, les variables d'entrée VAR1 du module MODULE1 et VAR2 du module MODULE2 ... prendront respectivement les valeurs

valeur1 et valeur2 ...

Il est donné la possibilité de spécifier des entrées périodiques. Exemple :

$T_k + T_p$ MODULE1(VAR3) = Valeur3 ;

Ceci signifie que la variable VAR3 du module MODULE1 est positionnée à la valeur Valeur3 à tous les instants $T_k + mT_p$ pour m = 0, 1, 2,

Une primitive T_s STOP peut être intercalée à tout point de la séquence et a pour effet de provoquer la suspension de la simulation à l'instant T .

III - 3.5. Interprétation

Toutes les mises à feu possibles à un instant donné sont effectuées simultanément. La détection de conflit (place pleine partagée par deux transitions susceptibles d'être mises à feu à cet instant) ou d'une situation de fonctionnement non sauf du RdPI (l'arrivée d'une marque dans une place coincide avec l'arrivée ou la présence d'une autre marque) entraine la suspension de la simulation et la sortie d'un message approprié.

Quand une marque arrive dans une place, elle déclenche l'exécution de la procédure associée. Cette exécution est effectuée en deux phases :

- à l'instant où elle est déclenchée, toutes les instructions précédant la primitive TEMP sont exécutées, puis le paramètre de la primitive TEMP est évalué et l'évènement correspondant est noté dans un échéancier global. La P.C est réexaminée au cas où cette exécution aurait rendu possible de nouvelles mises à feu ; toutefois, la marque dans la place est maintenue à l'état indisponible.

- à l'arrivée de l'évènement noté, les instructions qui suivent la primitive TEMP dans la procédure sont exécutées et la marque dans la place correspondante est mise à l'état disponible. La P.C est de nouveau examinée pour les mises à feu devenues possibles.

L'exécution des deux phases s'effectue avec les valeurs des variables prélevées au moment de l'activation de la procédure.

Si plusieurs procédures sont activées au même instant, les premières phases correspondantes sont exécutées avant le réexamen de la P.C. Il en est de même pour les secondes phases si l'arrivée des évènements correspondants se produit au même instant.

Dans le cas où l'évaluation du paramètre de la primitive TEMP conduit à une valeur nulle, la deuxième phase est exécutée derrière la première et la marque est portée tout de suite disponible. La présence de la primitive TEMP n'est pas obligatoire ; en son absence, la procédure est considérée comme ayant un temps d'exécution nul.

Les situations d'interférence dues à l'exécution parallèle des procédures ne sont pas détectées (problème de "déterminacy").

IV - EXEMPLE D'APPLICATION

Le langage est illustré en l'appliquant à la description d'un système multiprocesseur inspiré de celui défini par GOUNTANIS et VISS dans [E42] (figure VII.9)

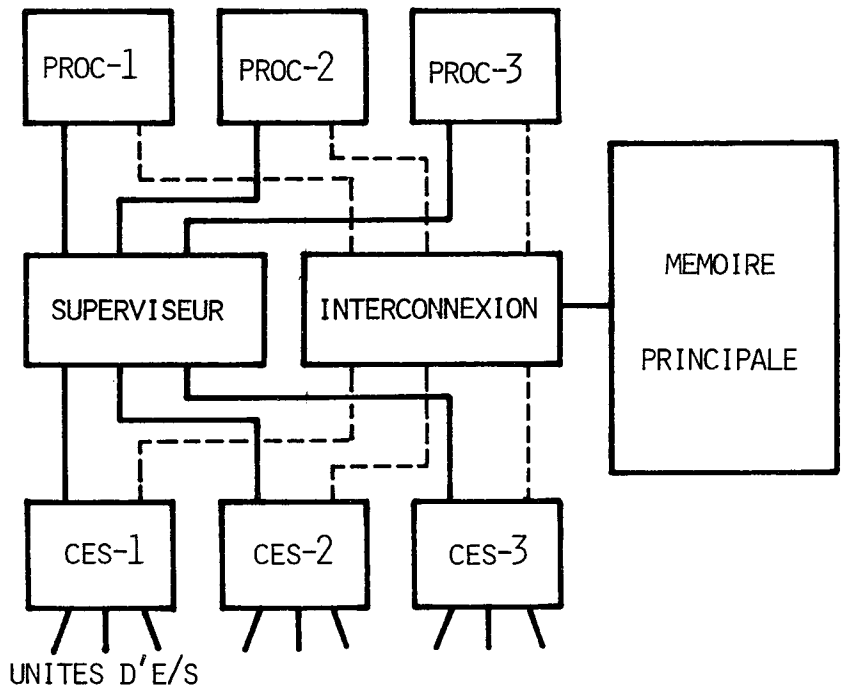


Figure VII-9

Des processeurs banalisés peuvent exécuter un certain nombre de tâches prédéfinies, stockées en mémoire principale. Ces tâches correspondent au traitement de demandes d'interruption positionnées par des unités d'entrées/sorties (UES). Le fonctionnement du système correspond à celui du contrôle

d'un processus industriel.

Chaque UES peut émettre une demande d'interruption (IT) vers le contrôleur d'entrées/sorties (CES) auquel elle est connectée,en indiquant le niveau de priorité de cette demande (IP) qui peut varier de 1 à 15. Le CES enregistre en mémoire le contexte associé à cette demande et la transmet au module SUPERVISEUR.

Le SUPERVISEUR gère l'activation des processeurs pour le traitement des IT émanant des CES.

Le module INTERCONNEXION gère les accès mémoire des processeurs et CES.

DESCRIPTION DES MODULES DU SYSTEME

α) Module PROCESSEUR

La figure VII.10 montre les principaux registres d'un module processeur ; l'algorithme de fonctionnement est illustré en figure VII.11

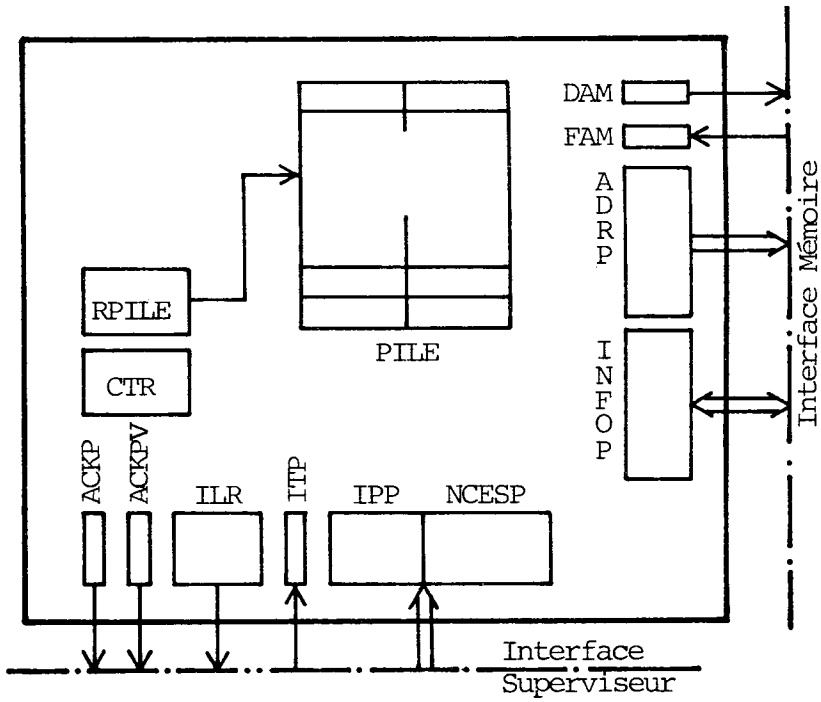


Figure VII-10

Structure d'un module PROCESSEUR

Chaque processeur dispose d'un registre ILR où il indique, à tout moment, son niveau d'interruptibilité. Ce niveau correspond à celui de l'IT traitée sauf quand le processeur entre dans une section critique ; dans ce cas le registre ILR est positionné à la valeur 15.

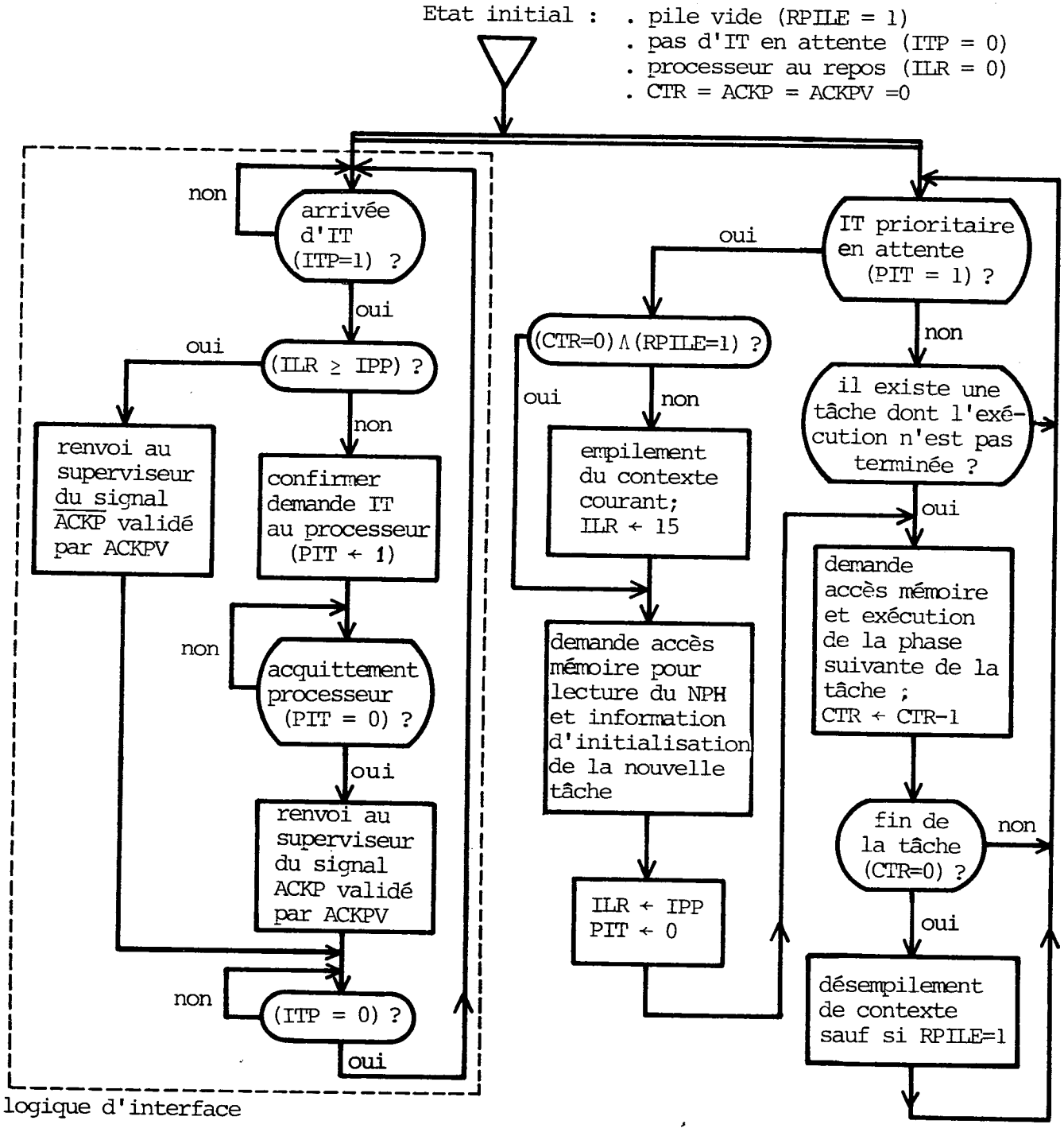


Figure VII-11

Organigramme de fonctionnement d'un module PROCESSEUR

Le SUPERVISEUR peut soumettre au processeur une nouvelle demande IT si la priorité IP de cette demande est supérieure à la valeur indiquée dans ILR. Il effectue au préalable une lecture du registre ILR et compare son contenu à la valeur IP. Si pendant ce temps, la valeur de ILR est restée inférieure à IP, le processeur interrompt la tâche en cours et effectue le changement de contexte nécessaire pour prendre en compte la nouvelle IT ; il acquitte alors le SUPERVISEUR en renvoyant le signal ACKP validé par ACKPV. Dans le cas contraire, le SUPERVISEUR reçoit le signal $\overline{\text{ACKP}}$.

Une tâche s'exécute par phases ininterruptibles (à la limite, une phase est une instruction). Le processeur ne peut donc prendre en compte une IT qu'à la fin de chaque phase. En aucun cas, le SUPERVISEUR ne peut forcer l'interruption immédiate du processeur. On suppose que la valeur ILR = 0 correspond à l'état du processeur au repos.

Remarque

Il se pose dans cette description le problème de la représentation des accès mémoire. Le processeur accède à la mémoire soit pour effectuer un changement de contexte, soit pour chercher les instructions et opérandes de la tâche en cours. Le nombre d'accès effectués lors d'un changement de contexte est constant ; le nombre d'accès effectués durant l'exécution de la tâche est variable. Tenir compte de la représentation de ces détails complique sans intêret l'exemple. On adoptera les simplifications suivantes :

- un changement de contexte nécessite un seul accès mémoire ;
- l'exécution d'une phase d'une tâche quelle qu'elle soit nécessite un seul accès mémoire ;
- le nombre de phases (NPH) pour chaque tâche est déterminé à partir du niveau de priorité de l'IT traitée et du numéro du CES qui l'a positionné. La correspondance (IP, NCES)<—>(NPH) peut être tabulée en mémoire comme le montre la figure VII.12.

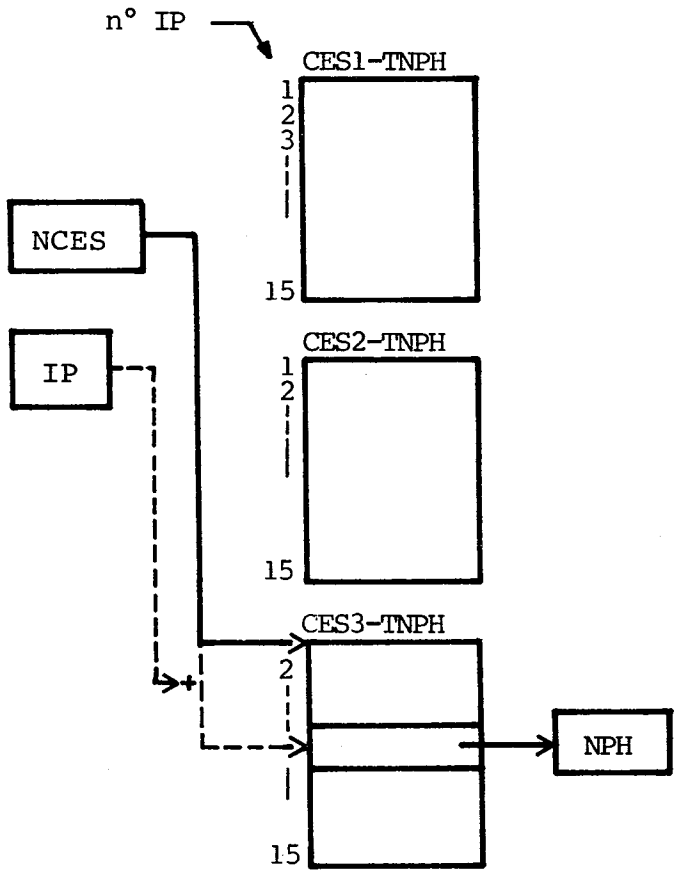


Figure VII-12

. La durée d'accès mémoire correspond au temps mis entre le positionnement d'une demande d'accès mémoire (DAM) et la réception du signal d'acquittement (FAM) renvoyé par le module INTERCONNEXION.

. Enfin, on suppose que chaque processeur gère, dans une mémoire locale, une pile dans laquelle il range son contexte chaque fois qu'il doit abandonner l'exécution d'une tâche au profit du traitement d'une IT plus prioritaire. Le changement de contexte correspond alors à l'empilement du contexte courant et à la lecture en mémoire du nouveau NPH et éventuellement d'autres informations concernant l'initialisation de la nouvelle tâche. La dernière tâche interrompue est reprise dès la fin de l'éxécution de la tâche en cours (sauf arrivée d'une IT prioritaire) ; dans ce cas, le changement de contexte ne nécessite aucun accès mémoire.

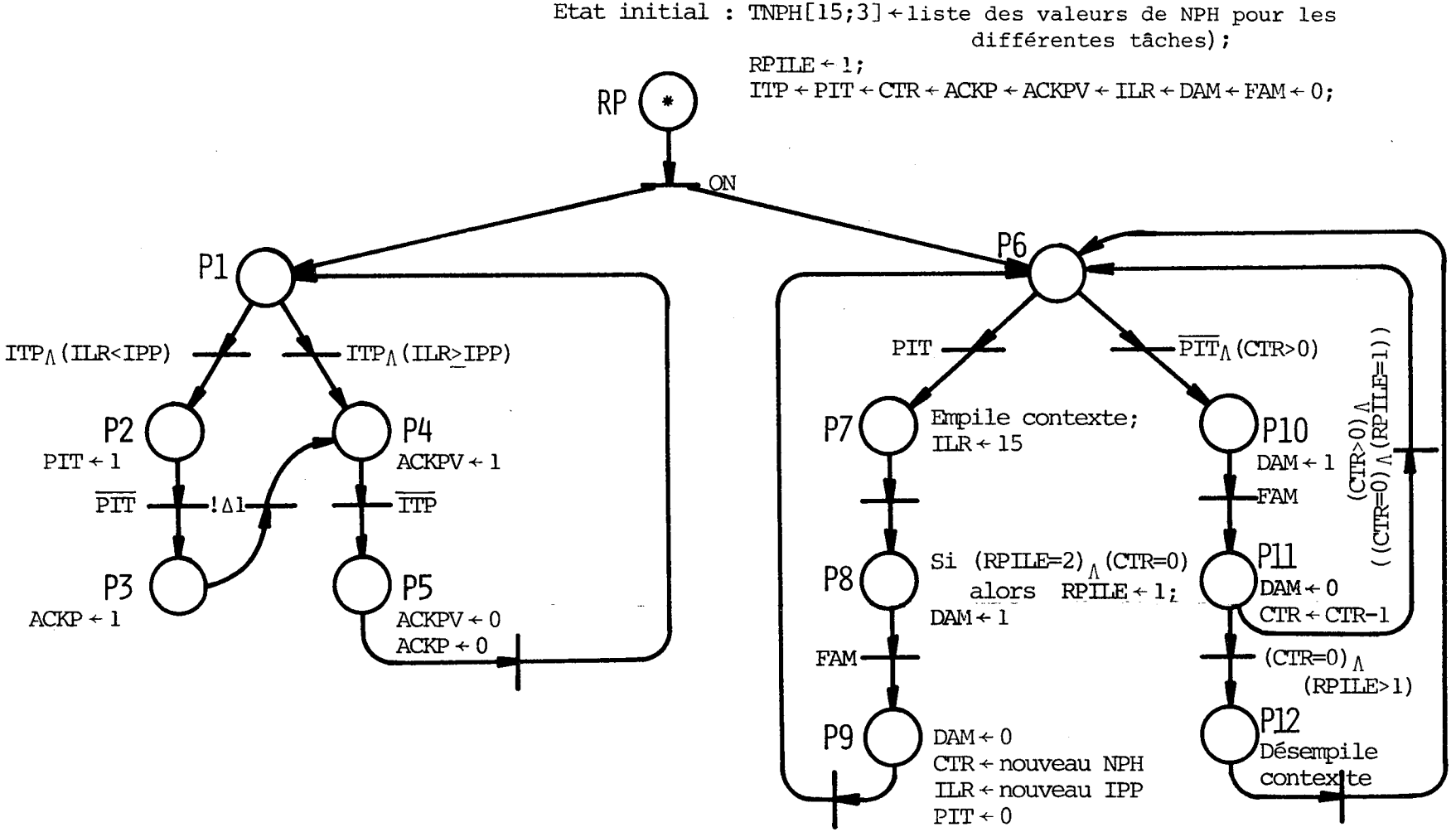


Figure VII-13

RdPI représentant un module PROCESSEUR

β) Module SUPERVISEUR

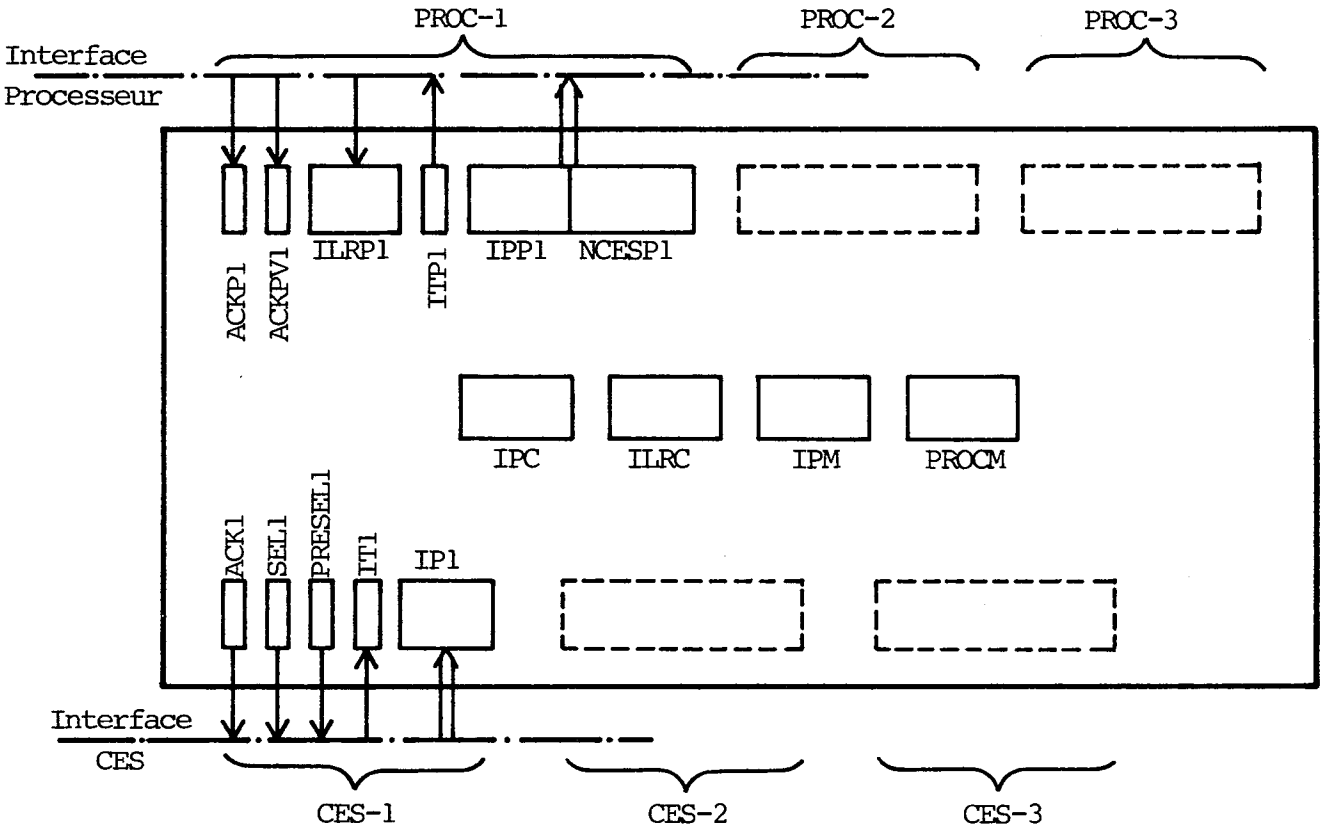


Figure VII-14

Structure du module SUPERVISEUR

Le SUPERVISEUR est activé sur apparition d'au moins une IT émanant de l'un des CESs qui lui sont connectés. Il effectue alors un travail en deux phases :

phase 1 : Sélection de l'IT la plus prioritaire.

Le SUPERVISEUR positionne un signal PRESEL à tous les CESs afin d'inhiber tout nouvel envoi de demande IT, puis après stabilisation, il sélectionne le CES ayant émis l'IT la plus prioritaire (IPC = plus grand IP présent). En cas de conflit, l'ordre physique des connexions partage entre les CESs. Le CES élu recevra le signal SEL et doit rester en connexion avec le SUPERVISEUR jusqu'à réception de l'acquittement du processeur qui traitera l'IT. Les

autres CESs recevront le signal $\overline{SEL}$ et sont libérés dès la remise à zéro du signal PRESEL.

phase 2 : Sélection d'un processeur pour le traitement de l'IT.

Le SUPERVISEUR fige les entrées ILR et sélectionne le processeur ayant le niveau d'interruptibilité le plus petit (ILRC = plus petit des ILRP). Deux cas peuvent se présenter :

a) IPC ≤ ILRC. Le traitement de l'IT est reporté et le SUPERVISEUR renvoie au CES sélecté le signal $\overline{ACK}$ validé par la remise à zéro de SEL

b) IPC > ILRC. Le SUPERVISEUR transmet alors la demande IT (avec le IP et le NCES) au processeur choisi et reste en attente pour transmettre au CES l'acquittement ACKP ou le refus $\overline{(ACKP)}$ qui sera rendu par celui-ci (voir fonctionnement d'un module processeur).

Un CES qui se voit refuser sa demande ($\overline{ACK}_{\wedge}\overline{SEL}$) doit la représenter pour le cycle suivant, mais si entre temps, de nouvelles demandes IT sont apparues, le CES doit en sélecter la plus prioritaire. De même, le SUPERVISEUR choisit de nouveau l'IT la plus prioritaire parmi celles qui sont présentes après nouvel envoi de PRESEL.

Il peut arriver qu'à la suite d'un premier échec, le SUPERVISEUR sélecte le même doublet (IPC, ILRC). Pour éviter un second échec, il réeffectue la phase b en mettant hors concours le processeur venant d'être élu.

Le RdPI représentant le module SUPERVISEUR est donné en figure VII.15.

γ) Module CES

La fonction du module CES est de scruter l'arrivée des ITs sur son interface UES. Il enregistre alors le contexte correspondant dans un mot réservé en mémoire (table TCES, figure VII.17) et transmet la demande au SUPERVISEUR.

Le niveau IP d'une IT positionnée par une UES peut varier de 1 à 15, mais à un moment donné, une seule demande IT peut être positionnée par une même UES. Tant que le signal PRESEL n'est pas présent, le CES a la possibilité de modifier une demande précédente faite au superviseur pour privilégier une nouvelle IT plus prioritaire.

Quand, après avoir été sélectionné ($\overline{PRESEL}_{\wedge}SEL$), le CES reçoit un acquittement positif ($ACK_{\wedge}\overline{SEL}$), il effectue un accès mémoire pour effacer de sa table TCES le mot mémoire correspondant au contexte de l'IT qui vient d'être acquitée.

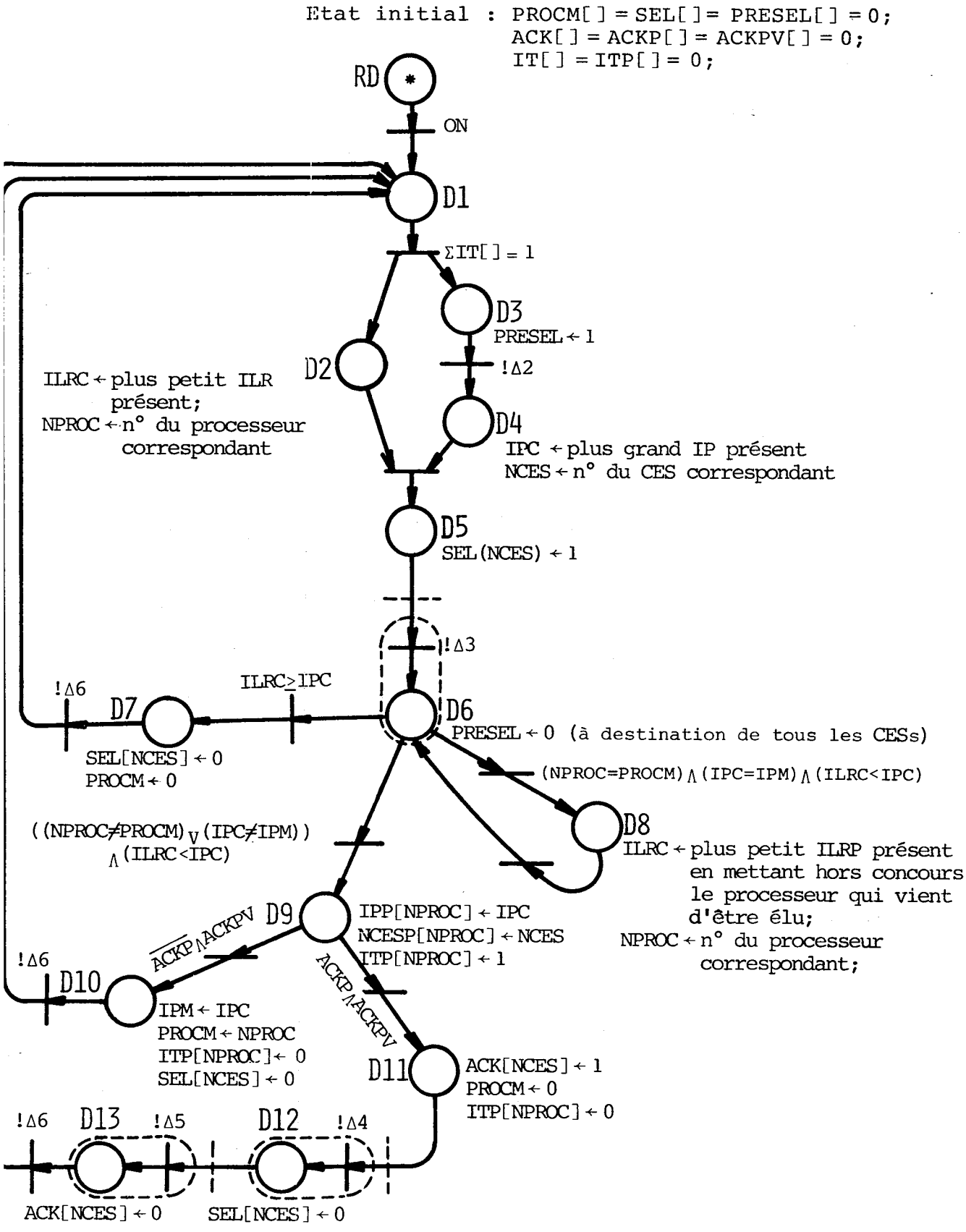


Figure VII-15

RdPI représentant le module SUPERVISEUR

ota : Les prédicats temporels !Δi sont utilisés ici pour représenter des temps de stabilisation de signaux. On peut vérifier, qu'étant données les règles d'interprétation décrites en III-3.5, ces temps peuvent être incorporés dans ceux des places de sortie.

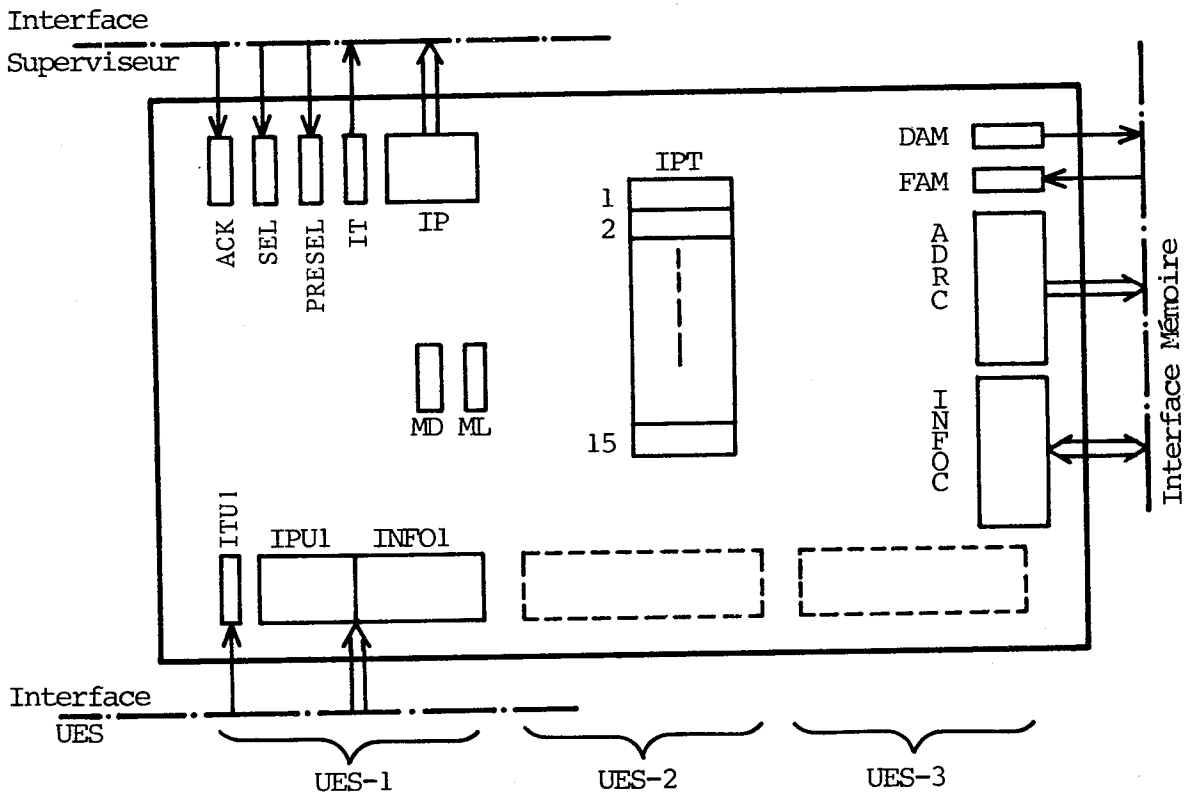


Figure VII-16

Structure d'un module CES

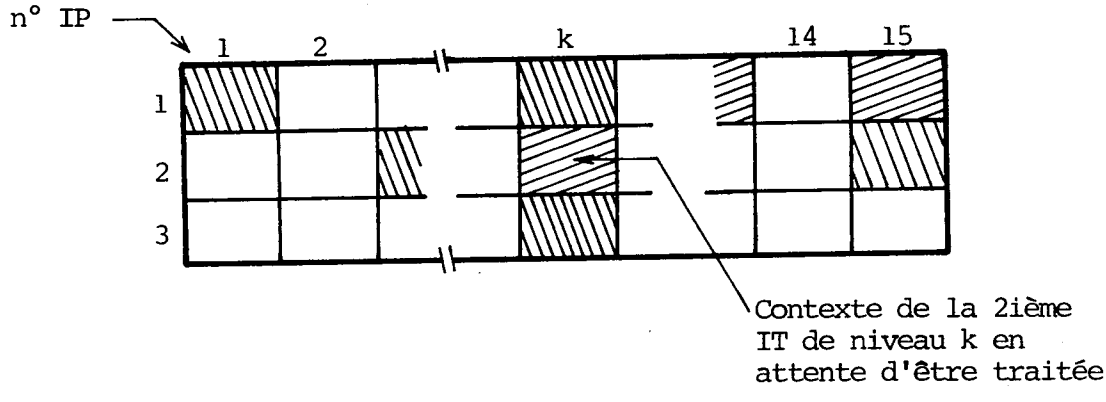
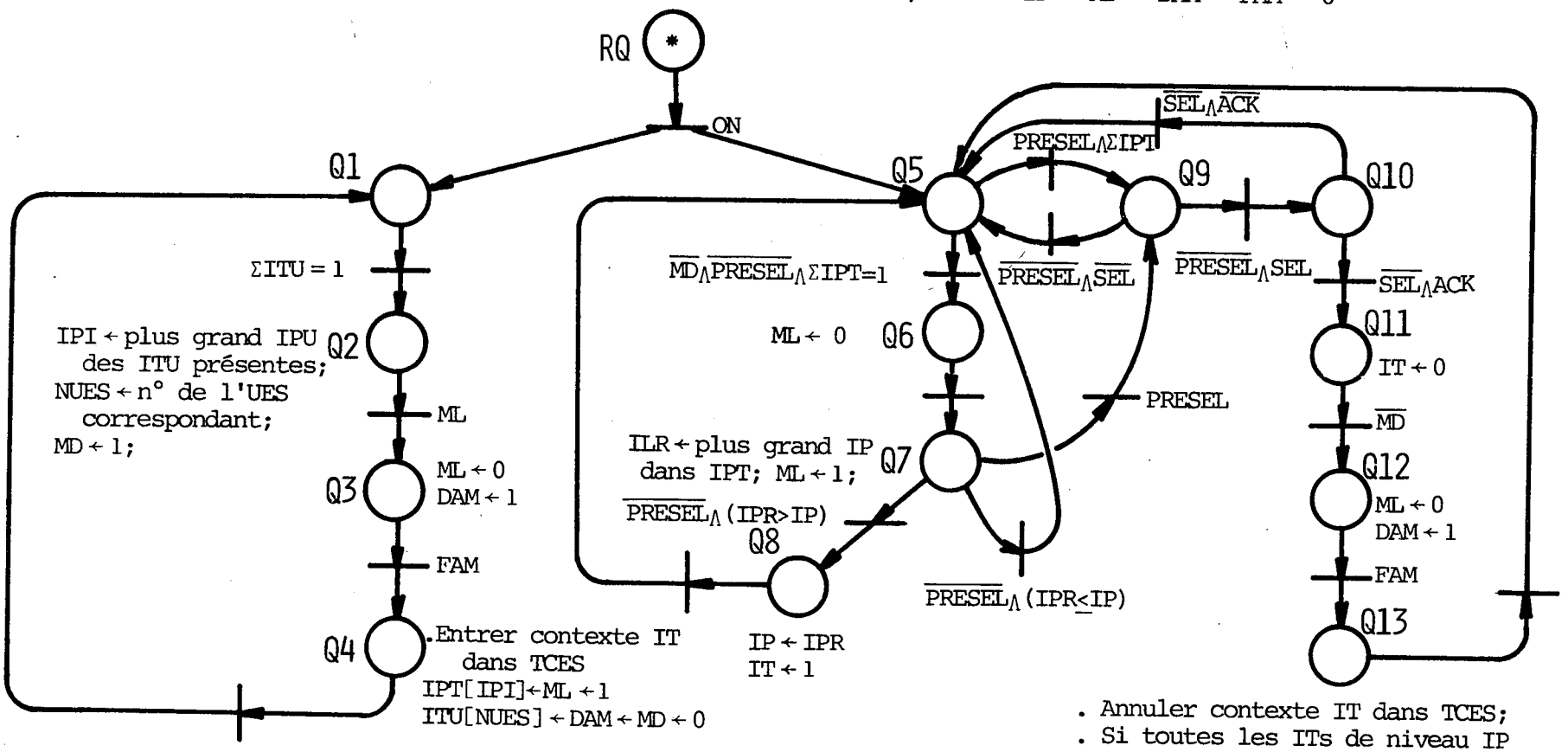


Figure VII-17

Table TCES pour le CESi

le RdPI représentant un module CES est donné en figure VII.18.

Le RdPI représentant un module CES est donné en figure VII.18.



Nota : MD et ML servent à synchroniser les accès mémoire et les lectures du tableau IPT effectués par les deux sous-modules.

Figure VII-18

RdPI représentant un module CES

δ) Module INTERCONNEXION

La structure de ce module est donnée en figure VII.19.

Les priorités d'accès mémoire sont fixées dans l'ordre décroissant CES-1 → CES-3 , PROC-1 → PROC-3 . On suppose que le cycle actif de la mémoire est de durée τ.

Le RdPI représentant ce module est donné en figure VII.20.

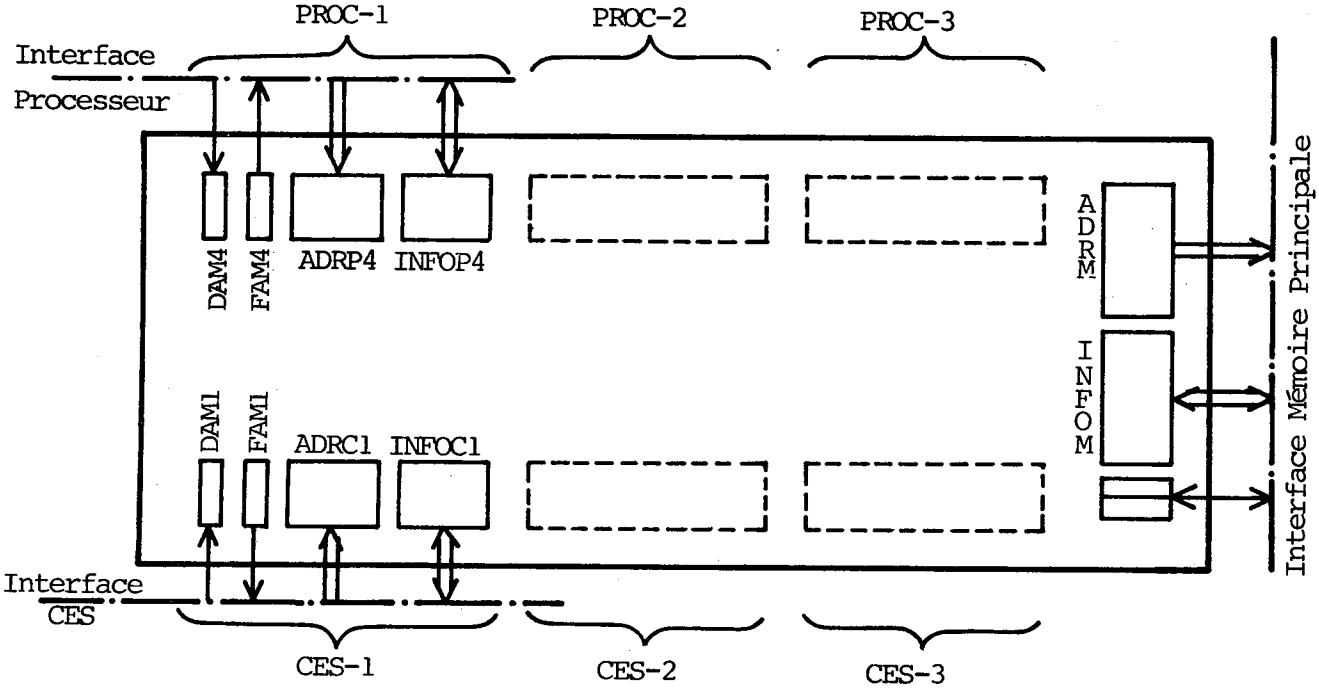


Figure VII-19

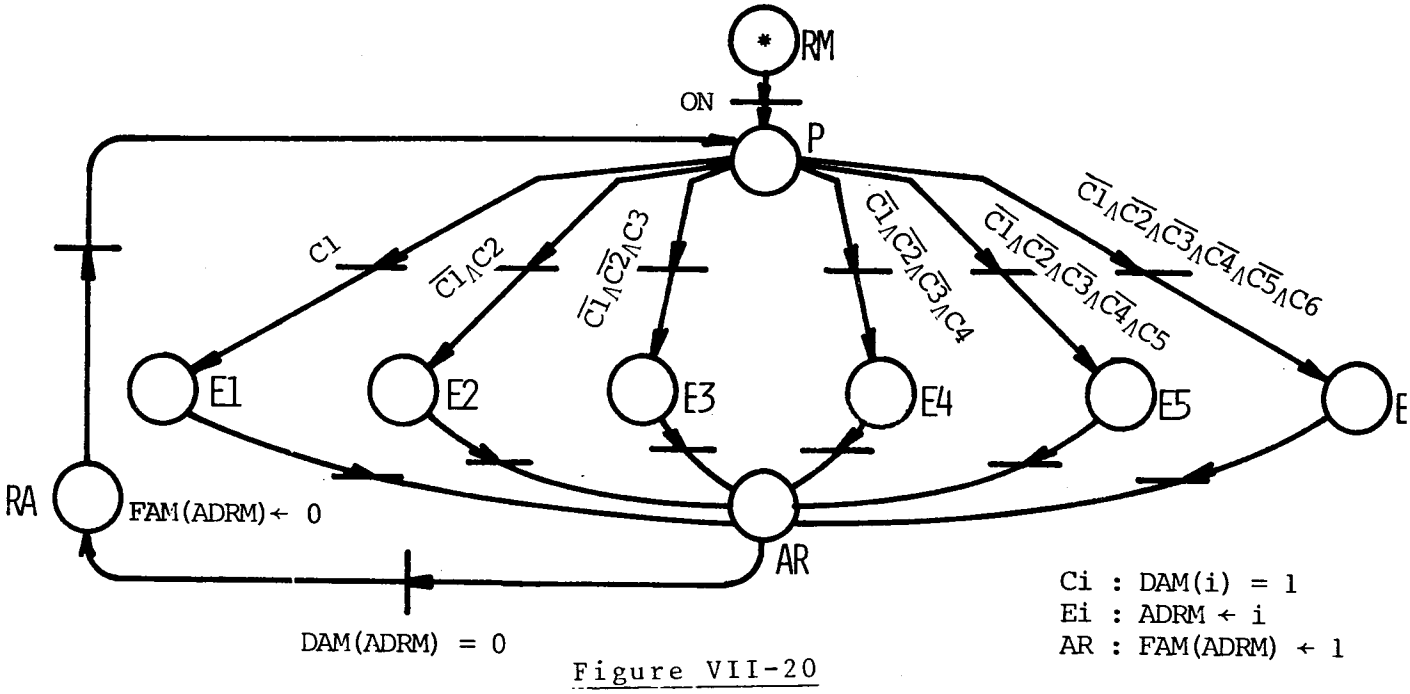


Figure VII-20

Le programme MAS de description du système global est donné en annexe.

V - RESEAUX DE PETRI GENERALISES ET MODELES DE FILES D'ATTENTE

V - 1. RESEAU DE PETRI GENERALISE : RdPG [E33] [E43]

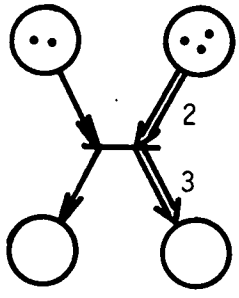
Définition 1 :

Un RdPG est un RdP tel que à chaque arc est associé un poids qui est une valeur entière positive. Une transition t_j est validée ssi chacune de ses places d'entrée contient au moins q marques, q étant le poids de l'arc reliant cette place à la transition t_j. Lors de la mise à feu d'une transition validée, le nombre de marques enlevées à ses places d'entrée et le nombre de marques ajoutées à ses places de sortie sont égaux aux poids des arcs respectifs.

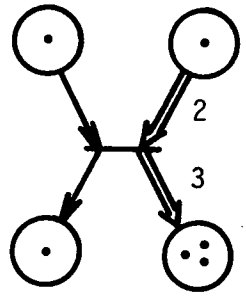
Représentation :

On représentera un RdPG de la même manière qu'un RdP en portant sur chaque arc le poids correspondant. On conviendra de représenter par un arc double tout arc dont le poids est supérieur à 1. Les arcs simples représentent des arcs de poids égal à 1.

Exemple



(a) avant mise à feu



(b) après mise à feu

V - 2. RESEAU DE PETRI GENERALISE AVEC TEST A ZERO : RdPØ [E28] [E30]

Définition 2 :

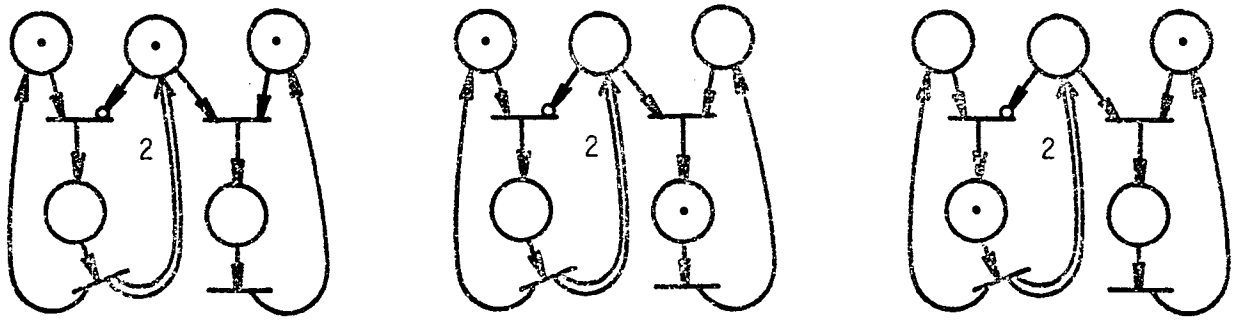
Un RdPG avec test à zéro est un RdPG dont les arcs entrants simples d'une transition peuvent être de deux types : direct ou complémenté . Les règles

d'évolution sont les mêmes que pour les RdPG sauf qu'une transition t_j n'est validée que si chaque place d'entrée reliée par un arc double ou simple-direct contient au moins un nombre de marques égal au poids de cet arc, et chaque place reliée par un arc complémenté est vide.

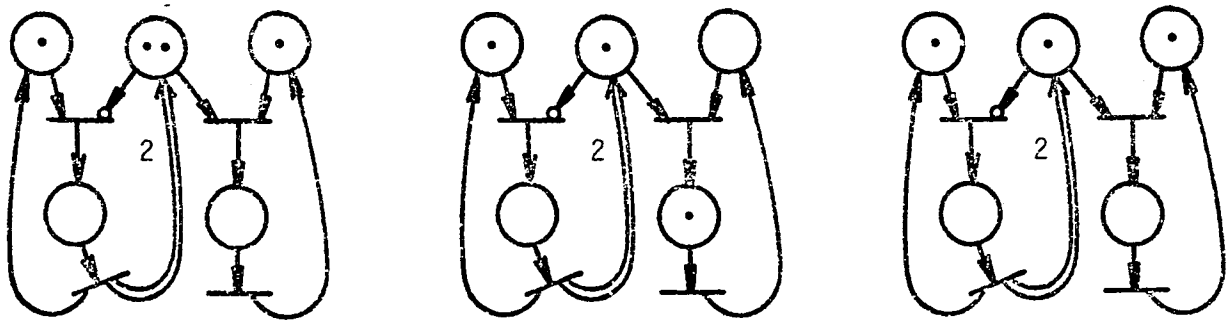
Représentation

On représentera un RdPØ de la même manière qu'un RdPG en distinguant les arcs complémentés en plaçant un petit cercle aux extrémités qui les connectent aux transitions.

Exemple



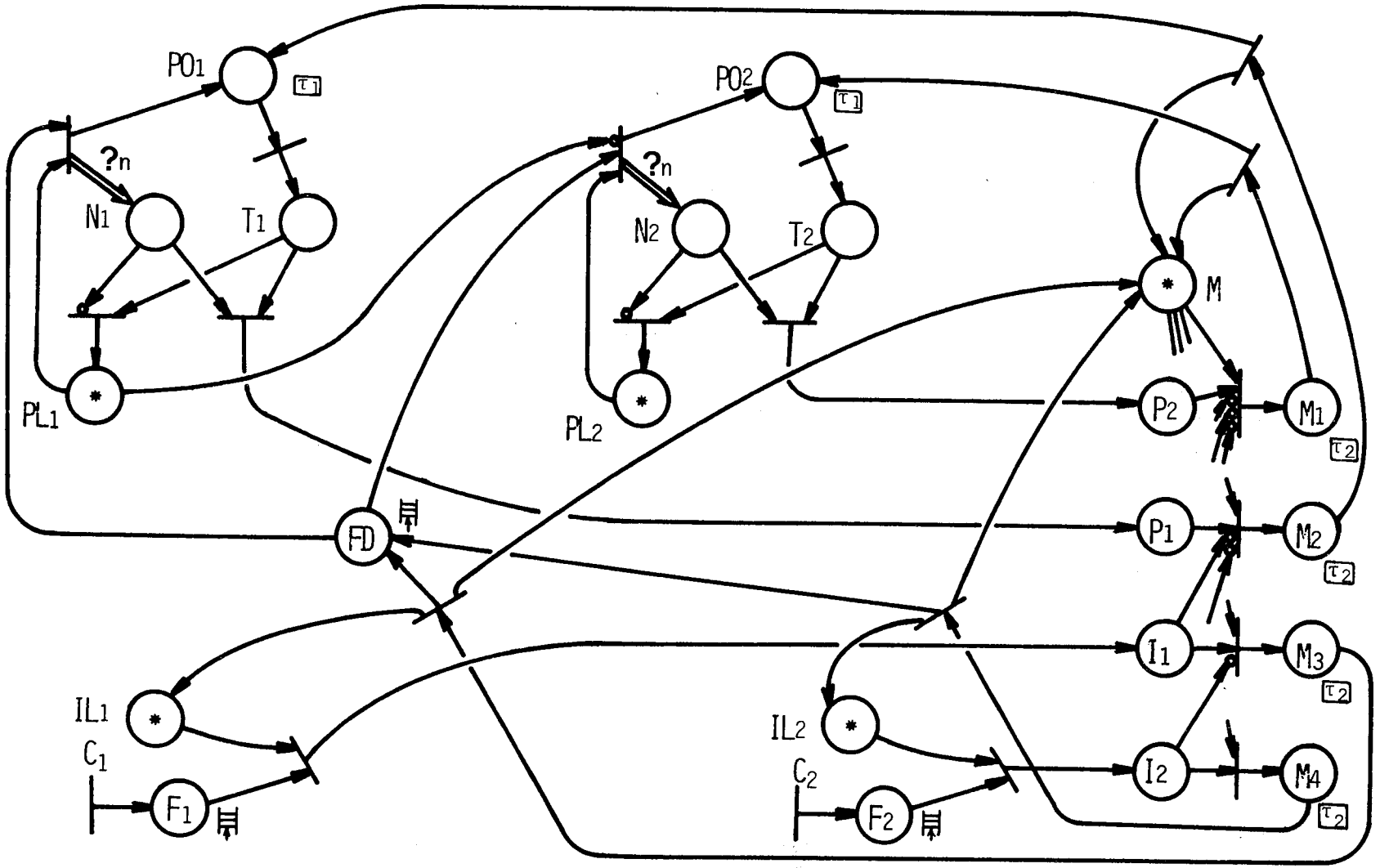
(a) (b) (c)



(d) (e) (f)

V - 3. EXTENSION DE M.A.S A LA DESCRIPTION DE MODELES DE FILES D'ATTENTE

On peut utiliser les RdPØ pour décrire le comportement d'un système



légende :

τ : place représentant une station d'exécution

: place représentant une file d'attente

?n : nombre aléatoire compris entre 1 et n

Figure VII-21

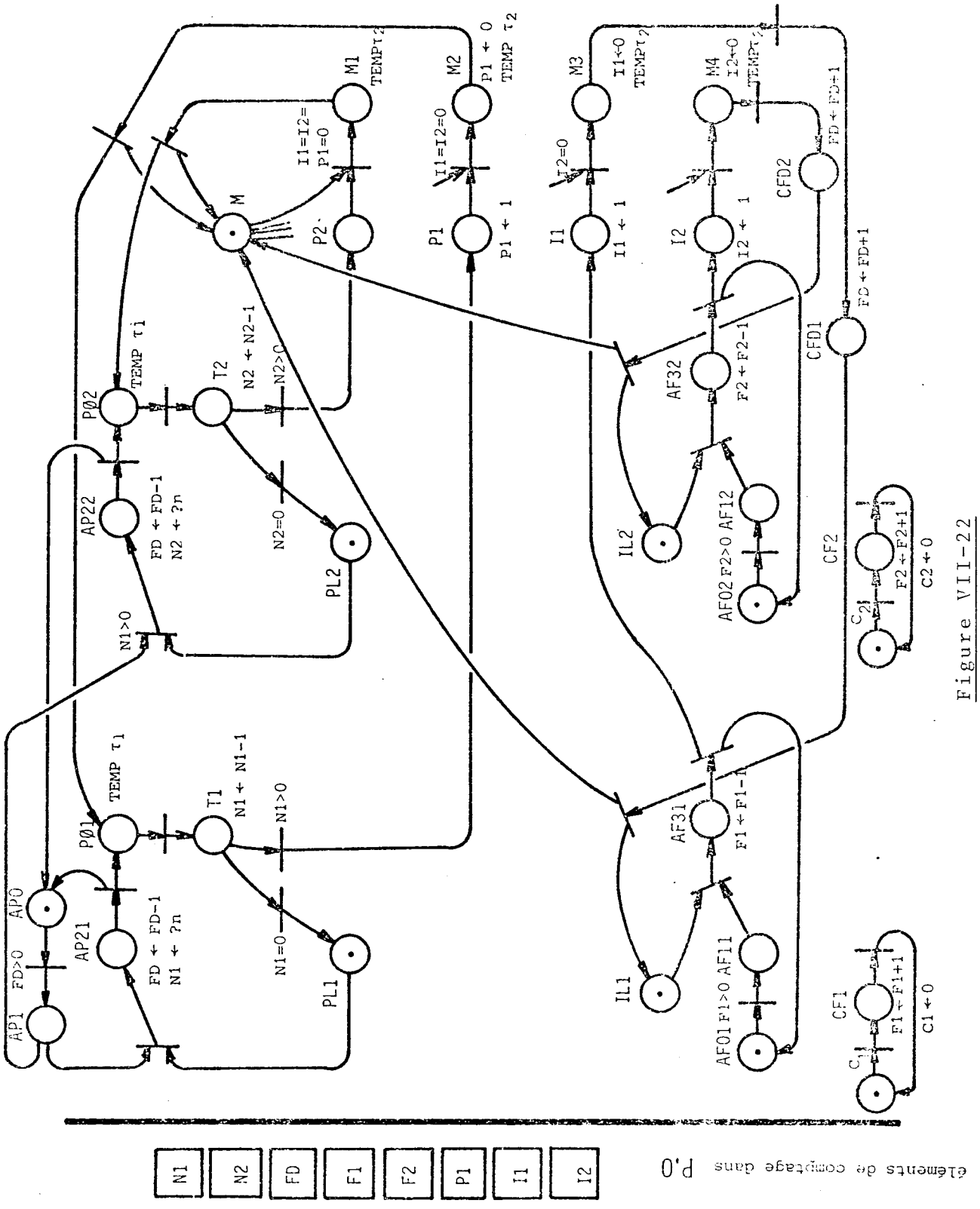


Figure VII-22

au niveau d'un modèle de file d'attente [E33] [E44]. De la même manière qu'on a défini les RdPT à partir des RdP, on peut définir les RdPØT à partir des RdPØ. Le modèle obtenu permet alors d'étudier par simulation, utilisant les primitives de M.A.S, le comportement du système à un niveau assez haut.

La figure VII.21 illustre l'utilisation des RdPØT à la description de l'exemple donné en IV pour le cas de deux processeurs et deux CES. On peut caractériser les places selon l'interprétation fonctionnelle qui leur est associée dans le modèle :

- station d'éxécution ($\boxed{\tau}$, où τ est le temps d'indisponibilité des marques dans cette place, ce qui correspond au temps d'exécution dans la station) ;
- file d'attente () ;
- place de synchronisation.

La figure VII.22 montre le modèle équivalent simulable à l'aide du langage M.A.S.

VI - CONCLUSIONS

L'intérêt de l'outil présenté est qu'il suggère une méthodologie de description fonctionnelle multiniveaux des systèmes ; ce qui permet de faire le pont entre une description de comportement et une description structurelle fine.

Le concepteur peut, en partant d'une description de haut niveau proche du cahier des charges, spécifier progressivement chaque module en éclatant les procédures dans la P.O en sous-procédures et en reportant leur séquencement dans la P.C.

Par ailleurs, l'utilisation des RdPI comme modèle mathématique de base offre un moyen puissant d'analyse. Cette analyse peut se faire de façon statique en étudiant les propriétés du RdPTS associé (sauf, vivant, persistant). D'autre part, en simulant le fonctionnement du système, on peut étudier son comportement dynamique, valider sa cohérence (taux d'occupation des modules, goulots d'étranglements aux interfaces) et mesurer ses perfor-

mances.

Certes, l'utilisation de cet outil est plus avantageuse pour des descriptions fonctionnelles d'un niveau assez haut.

Le choix des primitives du langage a été fait de façon à fournir un outil orienté utilisateur. Toutes les fonctions concernant aussi bien la synchronisation et la réalisation des liens entre modules restent transparentes à l'utilisateur et sont prises en charge par le système supportant la programmation dans ce langage.

Une première version du simulateur est actuellement en cours de mise au point [E45] . Certaines extensions sont envisagées, concernant particulièrement l'introduction de nouvelles primitives pour l'évaluation des performances. Ces primitives devront faciliter la génération des entrées de simulation et la collecte et analyse des résultats obtenus.

Conclusions

L'approche fonctionnelle dans le test des systèmes informatiques se justifie, car elle est indispensable, pour la vérification des dossiers en mise au point des prototypes et en intégration durant la fabrication de série. Sans cette vérification, l'emploi de tests optimisés élaborés à partir des dossiers matériels pour les vérifications en production et en maintenance n'est pas valide.

De plus, dans le cadre du test des sous-systèmes périphériques, comme dans celui des modules d'interface si les problèmes de testabilité n'ont pu être pris en compte à la conception, elle demeure la seule approche possible.

L'élaboration des tests à partir des dossiers matériels a fait l'objet de nombreuses recherches et on maitrise convenablement les méthodes permettant la création de profils de tests dans le cas des circuits logiques. En ce qui concerne l'approche fonctionnelle, il n'existe pas de méthodes algorithmiques permettant de systématiser l'élaboration de test et d'aboutir à une couverture exhaustive des fonctions du système. On a démontré que cette élaboration nécessite une modélisation du système selon des critères qui tiennent des aspects particuliers de l'implémentation et qui ont conduit à la définition des graphes cause-effet. A partir de cette modélisation, on a défini une méthodologie de construction des tests.

On a démontré, également, que l'application des tests fonctionnels nécessite un environnement particulier, compte tenu de la durée de la vérification et du respect des contraintes dynamiques qu'elle impose. Dans le cas particulier du test des sous-systèmes périphériques, et compte tenu des perspectives actuelles de conception de ces sous-systèmes, cet environnement peut être un interface utilisateur sous un système d'exploitation. Les tests d'acceptation, effectués après l'intégration des systèmes, nécessitent un logiciel autonome permettant d'avoir la maîtrise sur la génération des situations d'activité critique et de conflits.

Pour la phase de conception, on a proposé un outil de description et de simulation multiniveaux, assurant une continuité de la vérification depuis la traduction du cahier des charges, qui est une définition initiale du système jusqu'à l'implémentation finale.

Le graphe cause-effet peut être déduit simplement d'une description faite à l'aide de l'outil de conception. Ceci permet d'assurer la validité de la vérification de l'implémentation par rapport à la conception, quel que soit le niveau choisi. De plus, on pense intuitivement que cette déduction simple liée à l'aspect multiniveaux des deux représentations permet d'induire sur un ensemble de vérifications fonctionnelles définies à différents niveaux, la même relation de cohérence que celle garantie par l'utilisation de l'outil de conception.

ANNEXE - CHAPITRE VII

```
DCLMODTYPE PROCESSEUR(ACKP,ACKPV,ILR,ITP,IPP,NCESP,DAM,FAM)
   DCL PILE[15;2];
       RPILE←1;
       TNPH[15;3]← (liste des valeurs de NPH pour les différentes tâches);
       PIT;
   DCLPROC P2
       TEMP t_p2
       PIT←1
   FIN
   DCLPROC P3
       TEMP t_p3
       ACKP←1
   FIN
   DCLPROC P4
       TEMP t_p4
       ACKPV←1
   FIN
   DCLPROC P5
       TEMP t_p5
       ACKPV←ACKP←0
   FIN
   DCLPROC P7
       TEMP t_p7
       PILE[RPILE;]←CTR,ILR
       RPILE←RPILE+1
       ILR←15
   FIN
   DCLPROC P8
       TEMP t_p8
       RPILE←((RPILE=2)∧(CTR=0)+RPILE×((RPILE≠2)∨(CTR≠0))
       DAM←1
   FIN
   DCLPROC P9
       TEMP t_p9
       DAM←PIT←0
       CTR←TNPH[IPP;NCESP]
       ILR←IPP
   FIN
   DCLPROC P10
       TEMP t_p10
       DAM←1
   FIN
   DCLPROC P11
       TEMP t_p11
       DAM←0
       CTR←CTR-1
   FIN
   DCLPROC P12
       TEMP t_p12
       RPILE←RPILE-1
       CTR←PILE[RPILE;1]
       ILR←PILE[RPILE;2]
   FIN
```

```
    DCLCTRL
        RP  SI            ON                    ALORS P1,P6
        P1  SI        ITP∧(ILR<IPP)             ALORS P2
        P2  SI            ~PIT                  ALORS P3
        P3  SI            !Δ1                   ALORS P4
        P1  SI        ITP∧(ILR≥IPP)             ALORS P4
        P4  SI            ~ITP                  ALORS P5
        P5  SI             1                    ALORS P1
        P6  SI        PIT;(CTR>0)               ALORS P7;P10
        P7  SI             1                    ALORS P8
        P8  SI             FAM                  ALORS P9
        P9  SI             1                    ALORS P6
        P10 SI             FAM                  ALORS P11
        P11 SI        (CTR=0)∧(RPILE>1)         ALORS P12
        P11 SI  (CTR≠0)∨((CTR=0)∧(RPILE=1))     ALORS P6
        P12 SI             1                    ALORS P1
FINMOD

DCLMODTYPE SUPERVISEUR(ACK[3],SEL[3],PRESEL[3],IT[3],IP[3],ACKP[3],
                       ACKPV[3],ILRP[3],ITP[3],IPP[3],NCESP[3])
   DCL IPC,ILRC,IPM,PROCM,NPROC,NCES;
   DCLPROC D2
        TEMP t_d2
        ILRC←⌊/ILRP
        NPROC←ILRP⍳ILRC
   FIN
   DCLPROC D3
        TEMP t_d3
        PRESEL[]←1
   FIN
    D CLPROC D4
        TEMP t_d4
        IPC←⌈/IT×IP
        NCES←(IT×IP)⍳IPC
   FIN
    D CLPROC D5
        TEMP t_d5
        SEL[NCES]←1
   FIN
   DCLPROC D6
        TEMP t_d6+Δ3
        PRESEL[]←0
   FIN
    D CLPROC D7
        TEMP t_d7
        PROCM←SEL[NCES]←0
   FIN
   DCLPROC D8
        TEMP t_d8
        ILRC←⌊/ILRP[(NPROC≠⍳3)/⍳3]
        NPROC←ILRP[(NPROC≠⍳3)/⍳3]⍳ILRC
   FIN
```

```
    DCLPROC D9
        TEMP t_d9
        IPP[NPROC]←IPC
        NCESP[NPROC]←NCES
        ITP[NPROC]←1
    FIN
    DCLPROC D10
        TEMP t_d10
        PROCM←NPROC
        IPM←IPC
        ITP[NPROC]←SEL[NCES]←0
    FIN
    DCLPROC D11
        TEMP t_d11
        ACK[NCES]←1
        PROCM←ITP[NPROC]←0
    FIN
    DCLPROC D12
        TEMP t_d12+Δ4
        SEL[NCES]←0
    FIN
    DCLPROC D13
        TEMP t_d13+Δ5
        ACK[NCES]←0
    FIN
    DCLCTRL
        RD    SI          ON                  ALORS D1
        D1    SI        ∨/IT                  ALORS D2,D3
        D3    SI         !Δ2                  ALORS D4
        D2,D4 SI          1                   ALORS D5
        D5    SI          1                   ALORS D6
        D6    SI (ILRC≥IPC);((NPROC≠PROCM)∨(IPC≠IPM));1 ALORS D7;D9;D8
        D7    SI         !Δ6                  ALORS D1
        D8    SI          1                   ALORS D9
        D9    SI (~ACKP)∧ACKPV;ACKP∧ACKPV     ALORS D10;D11
        D10   SI         !Δ6                  ALORS D1
        D11   SI          1                   ALORS D12
        D12   SI          1                   ALORS D13
        D13   SI         !Δ6                  ALORS D1
FINMOD

DCLMODTYPE CONTROLEUR(ACK,SEL,PRESEL,IT,IP,DAM,FAM,ITU[3],IPU[3])
    DCL TCES[3;15];
        IPT[15],MD,ML,NUES,IPI,IPR;
    DCLPROC Q2
        TEMP t_q2
        IPI←⌈/ITU×IPU
        NUES←(ITU×IPU)\IPI
        MD←1
    FIN
    DCLPROC Q3
        TEMP t_q3
        ML←0
        DAM←1
    FIN
```

```
     DCLPROC Q4
          TEMP t_q4
          TCES[TCES[;IPI]⍳0;IPI]←NUES
          DAM←MD←ITU[NUES]←0
          IPT[IPI]←ML←1
     FIN
     DCLPROC Q6
          TEMP t_q6
          ML←0
     FIN
     DCLPROC Q7
          TEMP t_q7
          IPR←⌈/(IPT/⍳15).
          ML←1
     FIN
     DCLPROC Q8
          TEMP t_q8
          IP←IPR
          IT←1
     FIN
     DCLPROC Q11
          TEMP t_q11
          IT←0
     FIN
     DCLPROC Q12
          TEMP t_q12
          ML←0
          DAM←1
     FIN
     DCLPROC Q13
          TEMP t_q13
          TCES[;IP]←(¯1↓(1⌽TCES[;IP])),0
          IPT[IP]←(TCES[1;IP]>0)
          IP←DAM←0
          ML←0
     FIN
     DCLCTRL
          RQ  SI           ON                ALORS Q1,Q5
          Q1  SI          ∨/ITU              ALORS Q2
          Q2  SI           ML                ALORS Q3
          Q3  SI           FAM               ALORS Q4
          Q4  SI           1                 ALORS Q1
          Q5  SI (~MD)∧(~PRESEL)∧(∨/IPT)     ALORS Q6
          Q6  SI           1                 ALORS Q7
          Q7  SI PRESEL;(IPR>IP);(IPR≤IP)    ALORS Q9;Q8;Q5
          Q8  SI           1                 ALORS Q5
          Q5  SI PRESEL∧∨/IPT                ALORS Q9
          Q9  SI (~PRESEL)∧(~SEL)            ALORS Q5
          Q9  SI (~PRESEL)∧SEL               ALORS Q10
          Q10 SI (~SEL)∧(~ACK)               ALORS Q5
          Q10 SI (~SEL)∧ACK                  ALORS Q11
          Q11 SI          ~MD                ALORS Q12
          Q12 SI           FAM               ALORS Q13
          Q13 SI           1                 ALORS Q5
FINMOD
```

```
DCLMODTYPE INTERCONNEXION(DAM[6],FAM[6],ADRM)
   DCLPROC E(I)
        TEMP t_ei
        ADRM←I
   FIN
   DCLPROC AR
        TEMP t_ar
        FAM[ADRM]←1
   FIN
   DCLPROC RA
        TEMP t_ra
        FAM[ADRM]←0
   FIN
    DCLCTRL
        RM    SI            ON                                        ALORS P
        P     SI DAM[1];DAM[2];DAM[3];DAM[4];DAM[5];DAM[6]  ALORS E(I)
        E(I)  SI             1                                        ALORS AR
        AR    SI  (DAM[ADRM]=0)                                       ALORS RA
        RA    SI              1                                       ALORS P
FINMOD


DCLCONF
   CREE  PROC1,PROC2,PROC3:=PROCESSEUR;
         SUPERVISEUR1:=SUPERVISEUR;
         MEMOIRE:=INTERCONNEXION;
         CES1,CES2,CES3:=CONTROLEUR;
    LIE  PROC1(DAM)→MEMOIRE(DAM[4]);
         PROC1(ACKP,ACKPV,ILR)→SUPERVISEUR1(ACKP[1],ACKPV[1],ILRP[1]);
         MEMOIRE(FAM[4])→PROC1(FAM);
         SUPERVISEUR1(ITP[1],IPP[1],NCESP[1])→PROC1(ITP,IPP,NCESP);
             .
             .
             .
             .
             .
FIN


INIT
     PROC1(RP),PROC2(RP),PROC3(RP)=1;
     SUPERVISEUR.RD=1;
     CES1.RQ,CES2.RQ,CES3.RQ=1;
     MEMOIRE.RM=1;


  DONNEES
      T_1   CES1(ITU[1],IPU[1])=(1,1);
      T_2   CES1(ITU[2],IPU[2])=(1,5);
      T_3   CES2(ITU[2],IPU[2])=(1,8);
      T_4   CES3(ITU[2],IPU[2])=(1,13);
       .        .
       .        .
       .        .
      T_100 STOP
                .
                .
FIN
```

BIBLIOGRAPHIE

TPM

A-TPM-GST

[1] D.E. MULLER, "Evaluation of logical and organizational methods for improving the reliability and availability of a computer", Digest of the 1st Annual IEEE Comp. Conf., Sep. 67, pp. 53-55.

[2] J.J. DENT, "Diagnostic engineering requirements", Proc. AFIPS, SJCC, 1968, pp. 503-507.

[3] J.C. RAULT, "La détection et la localisation des défauts dans les circuits logiques", Digest of the 5th Int. Symp. on Fault Tolerant Computing (FTC-5), 1975, Paris, pp. 17-23.

[4] M. DEPEYROT, "Computer component and system testing as a part of computer-aided production and maintenance", Rapport LABORIA N° 83, Oct. 74, IRIA - Rocquencourt - FRANCE.

[5] E. DE ATLEY, "LSI testing is a large scale headache !", Electronic Design, Vol.16, Aug. 69, pp. 24-34.

[6] G.C. WARBURTON, "Automatic dynamic response system for testing semiconductors' Joint Conf. on Automatic Test Systems, IERE Conf. Proc., N° 17, April 70, pp. 467-484.

[7] H.H. HUANG, "MSI and LSI impact on digital systems testing", Proceedings of the 11th Annual Design Automation Workshop, 1974, pp. 159-165.

[8] D. DUMITRESCU, G. SAUCIER, "Test de mémoires dynamiques à technologie MOS", Digest of the 5th Int. Symp. on Fault Tolerant Computing (FTC-5), 1975, Paris, pp. 66-71.

[9] S.E. GROSSMAN, "Automated testing pays off for electronic system makers", Electronics, Sep. 19, 1974, pp. 96-109.

[10] M. BALL, F. HARDIE, "Effects and detection of intermittent failures in digital systems", AFIPS.FJCC, Vol. 35, 1969, pp. 329-335.

[11] A. AVIZIENIS, "Architecture of Fault-Tolerant Computing Systems", Digest of the 5th Int. Symp. on Fault-Tolerant Computing", 1975, pp. 3-16.

[12] H.J. BEUSHER and all., "Administration and Maintenance Plan of N° 2 ESS", BSTJ, Vol. 48, Oct. 69, pp. 2765-2815.

[13] Textes des journées d'études organisées par le LAAS/CNRS sur "les calculateurs numériques embarqués et leurs applications", Ed. Privat, 1974.

[14] P.J. SCOLA, "On-line computer-testing system", Second Workshop on Fault Detection and Diagnosis in Digital Systems, LEHIGH University, Dec. 71, pp. 103-114.

[15] B. COURTOIS, C. GUERIN, Ch. ROBACH, G. SAUCIER, A. VERDILLON, "Computer Test Program", Workshop on diagnosis and reliable design of digital systems, Pasadena, USA, Dec. 73.

[16] S.G. CHAPPELL and all., "LAMP-Logic-circuit simulators", The Bell System Tech. Journal, Vol. 53, n° 8, Oct. 74, pp. 1477-1503.

[17] M.A. BREUER, "General survey of design automation of digital computers", Proc. IEEE, Vol. 54, 1966, pp. 1708-1721.

[18] R.G. BENNETTS and D.W. LEWIN, "Fault diagnosis of digital systems - a review", COMPUTER, Vol. 12, July/August 1971.

[19] H.Y. CHANG, E. MANNING, G. METZE, "Fault Diagnosis of Digital Systems", Wiley-Interscience, New York, 1970.

[20] A.D. FRIEDMAN, P.R. MENON, "Fault Detection in Digital Circuits", Prentice Hall, 1971, Electrical Engineering Series.

[21] J.C. RAULT, "Bibliographie sur la détection et la localisation des défauts dans les circuits logiques", Rapport THOMSON-CSF, rue Vouillé, 75015 Paris, France.

[22] F.F. SELLERS, M.Y. HSIAO, L.W. BEARNSON, "Analysing errors with the Boolean-difference", IEEE Trans. on Comp. Vol. C-17, 1968, pp. 676-683.

[23] C.T. KU, G.M. MASSON, "The Boolean Difference and Multiple Fault Analysis", IEEE Trans. on Comp., Vol. C-24, N° 1, Jan. 75, pp. 62-72.

[24] A.C. PRIOR, R.G. BENNETTS, "Application of the Boolean-difference technique to sequential logic", Electronics Letters, Vol. 10, N° 23, Nov. 74, pp. 486-487.

[25] J.P. ROTH, "Diagnosis of automata failure : a calculus and a method", IBM J. Res. Develop, Vol. 10, 1966, pp. 278-291.

[26] D.B. ARMSTRONG, "On finding a nearly minimal set of fault detection tests for combinational logic nets", IEEE Trans. on Elec. Computers, Vol. EC-15, 1966, pp. 66-73.

[27] P.R. SCHNEIDER, "On the necessity to examine D-chains in diagnostic test generation - an example", IBM Journal, January 67, p. 114.

[28] J.P. ROTH, W.G. BOURICIUS, P.R. SCHNEIDER, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits", IEEE Trans. on Elec. Computers, Vol. EC-16, pp. 567-580.

[29] R.S. LEWIS, "An approach to test pattern generation for synchronous sequential networks", Ph.D Thesis, Southern Methodist University, 1967, USA.

[30] J. CAILLAT, "Contribution au test des circuits intégrés logiques", Thèse 3ème Cycle, ENSIMAG, Octobre 1976.

[31] T. ARIMA and all., "A new heuristic test generation algorithm for sequential circuits", 11th Design Automation Workshop, June 1974, pp. 169-176.

[32] S.G. CHAPPELL, "LAMP-Automatic test generation for asynchronous digital circuits", BSTJ, Vol. 53, n° 8, Oct. 1974, pp. 1477-1503.

[33] C.V. RAMAMOORTHY, "A structural theory of machine diagnosis", Proc. AFIPS, SJCC, 1967, pp. 743-756.

[34] C. TURCAT, A. VERDILLON, "Symétrie, récurrence et test", Digest of the 5th Int. Symp. on Fault-Tolerant Computing (FTC-5), 1975, Paris, pp. 56-60.

[35] S. SESHU, "On an improved diagnosis program", IEEE Trans. on Elec. Computers, Vol. EC-14, 1965, pp. 69-76.

[36] H.Y. CHANG, G.W. SMITH, R.B. WALFORD, "LAMP : System Description", The Bell System Tech. Journal, Vol. 53, N° 8, Oct. 74, pp. 1431-1449.

[37] E.W. THOMPSON, S.A. SZYGENDA, "Digital Logic Simulation in a Time-Based, Table-Driven Environment", COMPUTER, March 75, pp. 38-50.

[38] D.B. ARMSTRONG, "A deductive method for simulating faults in logic circuits", IEEE Trans. on Computers, Vol. C-21, N° 5, May 71, pp. 464-471.

[39] H.C. GODOY, R.E. VOGELSBERG, "Single pass error effect determination", IBM Technical Disclosure Bulletin, Vol. 13, April 71, pp. 3443-3444.

[40] R. DAVID, "Paradoxe du test des circuits combinatoires", Digital Processes, N° 1, 1975, pp. 333-336.

[41] Z. KOHAVI, "Switching and Finite Automata Theory", Ed. Mc Graw Hill, Computer Science series, 1970.

[42] C. HENNIE, "Finite-state models for logical machines", Ed. John Wiley & Sons, 1968.

[43] R. DAVID, G. BLANCHET, "Sur la détection des pannes dans les circuits combinatoires par des séquences d'entrée aléatoires", 5ème Colloque Int. sur les calculateurs tolérant les pannes (FTC-5), Juin 75, p. 241.

[44] J.J. SHEDLETSKY, E.J. McCLUSKEY, "The error latency of a fault in a combinational digital circuit", IEEE Trans. on Computers, Vol. C-25, N° 6, June 76, pp. 655-659.

[45] V. AGRAWAL, P. AGRAWAL, "An automatic test generation system for ILLIAC IV logic boards", IEEE Trans. on Comp., Sep. 72, pp. 1015-1017.

[46] M.A. BREUER, "The Effects of Races, Delays and Delays Faults on Test Generation", IEEE Trans. on Comp., Vol. C-23, N° 10, Oct. 74, pp. 1078-1092.

[47] J. LEBRUN, Ch. ROBACH, G. SAUCIER, "Processor testability and design consequences", IEEE Trans. on Comp., June 1976.

[48] H.Y. CHANG, G.W. HEIMBIGNER, "LAMP : Controllability, Observability and Maintenance Engineering Technique (COMET)", BSTJ, Vol. 53, N° 8, Oct. 74, pp. 1505-1534.

[49] "Automatic testing", Electronic Engineering, August 73, pp. 37-40.

[50] W.L. ALLAIN, "Integrated instrument setup tests intricate assemblies", Electronics, Oct. 9, 1972, pp. 104-109.

[51] M.J. RIEZENMAN, "Improved fault-finding with automatic functional testers", Electronics, August 14, 1972, pp. 115-123.

[52] A. TOTH, C. HOLT, "Automated Data Base-Driven Digital Testing", COMPUTER, January 1974, pp. 13-19.

[53] "Functional Characteristics of the X-Processors on the Machine Language Level", Carnet de notes CII-GX-B-02-01.

[54] J.P. RIGAULT, "Utilisation de registres à décalage pour le test et diagnostic des systèmes digitaux", Séminaire ENSIMAG, Mai 1974, Grenoble, FRANCE.

B - TPM - TMI

[1] T.J. CHANEY, C.E.MOLNAR, "Anomalous Behavior of Synchronizer and Arbiter circuits", IEEE Trans. on Comp., April 1973, pp 421-422.

[2] G.R. COURANZ, D.F. WANN, "Theoretical and Experimental Behavior of Synchronizers Operating in the Metastable Region", IEE Trans. on Comp., vol. C-24, N°6, june 1975, pp 604-614.

[3] I. CATT, "Time Loss Through Gating of Asynchronous Logic Signal Pulses", IEEE Trans. on Elec. Comp., vol. EC-15, february 1966, pp 108-111.

[4] P. CORSINI, "n-User Asynchronous Arbiter", Electronics Letters, 9 th, vol.11, N°1, January 1975.

[5] W.W. PLUMMER, "Asynchronous Arbiters", IEEE Trans. on Comp., vol C-21, N°1, january 1972, pp 37-42.

[6] R.C. PEARCE, J.A. FIELD, W.D. LITTLE, "Asynchronous Arbiter Module" IEEE Trans. on Comp., september 1975, pp 931-935.

[7] R. DAVID, B. DESCOTES-GENON, "Arbitres Asynchrones", Laboratoire d'Automatique de Grenoble (LAG) - France, Publication interne, sept.1975.

[8] M.A. BREUER, R.L. HARRISON, "Procedures for Eliminating Static and Dynamic Hazards in Test Generation", IEEE Trans. on Comp., vol. C-23, N°10, october 1974, pp 1069-1078.

[9] J. SIFAKIS, "Modèles Temporels des Systèmes Logiques", Thèse Doc.ing., Université de Grenoble, mars 1974.

[10] F.J. HILL, G.R. PETERSON, "Digital Systems: Hardware Organization and Design", Ed. John Wiley & sons, 1973, pp 270-297.

C - TPM - TSP

[1] JEFFREY P. BUZEN, "I/O Subsystem Architecture", Proc. of the IEEE, vol.63, N°6, june 1975, pp 871-879.

[2] A.S. HOAGLAND, "Mass Storage - Past, Present and Future", AFIPS, Proc. FJCC, vol 41, 1972, pp 985-991.

[3] G.W. NELSON, "OPTS-600 On-line Peripheral Test System", AFIPS Proc. FJCC, vol 33, 1968, pp 45-50.

[4] WALLNER, ARTHUR, "Error Detection for Peripheral Storage Devices" Computer Design, Jan. 1972, p.57.

[5] V. FARMER, "IBM 3850 extends VS to tape cartridge" Computer World vol.8, pp 1-5, october 1974.

[6] T. ATKINSON, "Architecture of series 60/level 64", Honeywell Computer Journal, vol.8, july 1974, pp 94-106.

[7] H. CHANG, G. HEIMBIGNER, D. SENESE, T. SMITH, "Maintenance Techniques of a Microprogrammed Self-Checking Control Complex of an Electronic Switching System", IEEE Trans. on Comp., vol C.22, n°5, may 1973 pp 501-512.

[8] R. SUDA, "An application-oriented multiprocessing system, Part V: The Diagnostic Monitor", IBM Sys. Journal, vol.6, n°2, 1967, pp 116-123.

[9] G.R. AHEARN, Y. DISHON, R.N. SNIVELY, "Design innovations of the IBM 3830 and 2835 storage control units", IBM.J.Res.Develop., vol.16, january 1972, pp 11-18.

[10] STC - Coupleur BM3000 - 73341 - 73342.

[11] J.W. IRWIN, J.V. CASSIE, H.C. OPPEBOEN, "The IBM 3803/3420 Magnetic Tape Subsystem", IBM J.Res.Develop., Sept. 1971, pp. 391-399.

[12] M. ROTTIER, "Note d'information générale sur les problèmes de maintenabilité", Honeywell-Bull France, document interne n° 69/22.300-833 - décembre 1969.

[13] W.R.ELMENDORF, "Cause-effect Graphs in Functional Testing", Int.IBM report, TR 00.2487, november 1973.

[14] IBM SYSTEM/360 OPERATING SYSTEM, "On-Line Test Executive Program" OLTEP SRL (Release 20-1), N° GC 28-6650.

[15] CII - ORDINATEUR 10070, Bibliothèque des programmes de test, "DCP: Diagnostic Control Program et DPM: Diagnostic Program Monitor pour DIMAS, DIAD, VISU, IMP 72444 et 72445, CR/CP, URM 1600 Bpi, COCE et KSR".

[16] CII - DOC/DFQ/TD MA/CL,"Programme de test des unités disque DIMAS, MD25 MD50 sur IRIS 45/60" Section SPD N°8, version 01, mai 1973.

[17] D.L.DROULETTE, "Recovery Through Programming System/360 - System/370" AFIPS - FJCC - vol.38, 1971, pp 467-476.

[18] D.A. LAWRENZ, "Diagnostic test design", Digest of the 3th Int.symp. on Fault-Tolerant computing, 1973, pp 5/8-11.

[19] P. DECITRE, M. DEPEYROT, A. JORRY, M. MOALLA, "The COSU Through-Line Test Process Interpreter", 5th Int. Symp. on Fault-Tolerant Computing, June 1975, Paris (et rapport de recherche IRIA-LABORIA n°95, janvier 1975).

[20] A. JORRY, M. MOALLA, "Programme conversationnel de test hardware des périphériques", rapport de projet ENSIMAG, juin 1973.

[21] A. JORRY, M. MOALLA, J. THIBAULT, G. VIGREUX, "COSU, Programme Conversationnel de test Sous-Utilisateur des périphériques", rapport de projet (seconde version), IRIA-CETASI-IE/51, juillet 1974.

[22] A. JORRY, M.MOALLA, "Manuel utilisateur de COSU sous SIRIS 7/8", IRIA-CETASI, IP/217, juillet 1974.

[23] C.I.I. - Manuel d'utilisation, "Procedures systemes sous SIRIS 7/8", (v B08A/C01A), réf. 3642 E3/F3, novembre 1972

[24] C.I.I. - Manuel d'utilisation "Système de Gestion des Transmissions (SGT) sous SIRIS 7/8", réf. 3850 E3/F3, février 1973.

[25] C.I.I. - Manuel d'utilisation et d'opération "Système de Gestion de Fichiers (SGF) sous SIRIS 7/8", réf. 3704 E1/FR, décembre 1971.

D - TPM - TVF

[1] J. KUNTZMANN, "Théorie des Réseaux", Ed. DUNOD, Paris, 1972

[2] J.L. BAER, D.P. BOUET, G. ESTRIN, "Legality and Other Properties of Graph Models of Computations", Journal of the ACM, vol.17, N°3, july 1970, pp 543-554.

[3] M.R. PAIGE and al, "Structural Techniques of Program Validation" COMPCON IEEE Comp. Society Int. Conference, San Francisco, Feb. 1974.

[4] M.E. FAGAN, "Design and Code Inspections and Process Control in the Development of Programs", IBM system Dev. Division, Kingston, N.Y. 12041 TR 21 572, january 1974.

[5] D.G. BRITTLE, "Notation Describing Fault-Related Behavior of Logic Modules", Electronics letters, 30th, vol.10, N°11, may 1974, pp.215-216.

[6] D.A. LAWRENZ "Diagnostic Test Design", 4th Int. Symposium on Fault-Tolerant Computing (FTC-4), june 1974, pp 5-8.

[7] CII / DFQ / TP, "Spécification de définition du programme SIMUX" Projet Gamme X, carnet de notes réf. 00.801, novembre 1974.

[8] CII / DOC / DFQ / TD MA/CL, "Programmes de test des DIMAS MD25/MD50 sur IRIS 45/60", SPD N°8, version 01, mai 1973.

[9] CII / DOC / DFQ / TD, "Programme 6-PERIPH: Test des E/S simultanées et mise en évidence des défauts apparaissant en exploitation sous SIRIS 6" (document non référencé).

[10] CII / DOC / DFQ / TD, "Programme TESI de test du fonctionnement des UEM en simultanéité sur IRIS 60", SPR N°112, version 03, octobre 1973.

[11] CII / DOC / DFQ / TD MD/CD, "Présentation du SYSSWAP 3-5 sur IRIS 80-10070", SPD N°46, version 01, octobre 1973.

[12] CII / DOC / DFQ / TD MT/MF, "Programmes de test de simultanéité TESU-TESI-CESI et ECSI sur IRIS 80", SPR N°5, version 2, juin 1973.

[13] IBM-MDP, "ASCP- Automatic System Checkout Program", SYS370 - ASCP user's guide, D99 - 0370A - 00, March 1972.

[14] M. MOALLA, "Tests de Simultanéité en Intégration des Systèmes", Séminaire ENSIMAG, Grenoble, juin 1975.

[15] D.L. PARNAS, "On the Criteria to be used in Decomposing Systems into Modules", Com. of the ACM, vol.15, N°12, december 1972, pp 1053-1058.

E-VFC-MAS

[1] R.I. GARDNER, "A Methodology for Digital System Design Based on Structural and Functional Modeling", UCLA-ENG-7488, January 1975.

[2] R.I. GARDNER, G. ESTRIN, H. POTASH, "A Structural Modeling Language for Architecture of Computer Systems", Int. Symp. on Comp. Hardware Description Languages and Their Applications, New York, Sept. 1975.

[3] YAOHAN CHU, "Why Do We Need Computer Hardware Description Languages ?", Computer, Dec. 1974, pp. 18-22.

[4] D.L. PARNAS, J.A. DARRINGER, "SODAS and a methodology for system design", FJCC, 1967, pp. 449-474.

[5] J.L. BAER, "Models for the Design, Simulation and Performance of Distributed-Function Architecture", Computer, March 74, pp. 25-30.

[6] B. COURTOIS, "On Balancing Safety and Reliability of Hybrid and Bi-Duplexed Systems", 6th Int. Symp. on Fault-Tolerant Computing, Pittsburgh, June 76, pp. 52-57.

[7] M. MOALLA, J. SIFAKIS, "A Design Methodology for Complex Logical Systems", 2nd International Symp. on Discrete Systems, Dresden, March 1977.

[8] M.R. BARBACCI, "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems", IEEE Trans. on Comp., Vol. C-24, N° 2, Fév. 1975, pp. 137-150.

[9] M.R. BARBACCI, "Introduction Computer Hardware Description Languages", Survey, Computer, Déc. 1974, pp. 27-51.

[10] J.J. CLANCY, M.S. FINEBERG, "Digital Simulation Languages : A critical and Guide", Proceedings FJCC, 1965, pp. 23-36.

[11] F. ANCEAU, P. LIDDEL, J. MERMET, C. PAYAN, "CASSANDRE : A Language to Describe Digital Systems, Applications to Logic Design", 3rd Int. Symp. Comp. and Inf. Science, Miami, Flo., Déc. 1969.

[12] J.R. DULEY, "DDL - A Digital Design Language", Ph.D. Dissertation, Dep. El. Eng., Univ. Wisconsin, Madison, June 1970.

[13] Y. CHU, "Introducing Computer Design Language", Digest of papers, Compcon 72, San Francisco, Sep. 1972, pp. 215-218.

[14] T.J. SCHRIBER, "Simulation using GPSS", John Wiley and Sons, 1974.

[15] O.J. DAHL, K. NYGAARD, "SIMULA : an Algol Based Simulation Language", Comm. ACM, Vol. 9, Sept. 1966.

[16] P.J. KIVAT, R.E. VILLAVUEVA, H.M. MARKOWITZ, "The SIMSCRIPT II Programming Language", Prentice Hall, Englewood Cliffs, N.J. 1969.

[17] D. POTIER, "La modélisation des Systèmes Informatiques", Bulletin de Liaison de l'IRIA, Janvier 1974, pp. 2-12.

[18] F.J. HILL, "Updating AHPL", Int. Symp. on Comp. Hardware Description Languages and Their Applications, New York, Sept. 1975.

[19] F.J. RAMMING, "DIGITEST II : An Integrated Structural and Behavioral Language", Int. Symp. on Comp. Hardware Description Languages and their Applications, New York, Sept. 1975.

[20] V.M. GLUSHKOV and A.A. LETICHEVSKII, "Theory of Algorithms and Discrete Processors", Advances in information Systems Science, Vol. 1, Chap. I, pp. 1-58, Ed. Julius T. TOU.

[21] K.A. DUKE, H.D. SCHNURMANN, T.I. WILSON, "System Validation by Three-Level Modeling Synthesis", IBM J. RES. DEVELOP., March 1971, pp. 166-174.

[22] J. LEROUDIER, M. PARENT, "Quelques aspects de la modélisation des systèmes informatiques par simulation à évènements discrets", RAIRO Informatique, Vol. 10, n° 1, Janvier 1976, pp. 5-26.

[23] A.N. HABERMAN, "Synchronization of Communicating Processes", Comm. ACM, Vol. 15, N°3, 1972, pp. 171-176.

[24] N. WIRTH, "Algorithms + Data Structures = Programs", Prentice Hall, Inc., Englewood Cliffs, N.J. 1976, p. 366.

[25] P. ROBERT, J.P. VERJUS, "Towards Autonomous Descriptions of Synchronization Modules", Rapport de Recherche ENSIMAG, RR n°47, Oct. 76.

[26] C.A. PETRI, "Communication with automata", Technical Rep. n° RADC-TR-65-377, Vol. 1, Rome Air Develop. Center, Griffis Air Force Base, New-York, Jan. 1966. (Traduit de "Kommunikation mit Automaten", Univ. de Bonn, 1962).

[27] A.W. HOLT, "Information System Theory Project", Applied Data Research, Incorporated Princeton, New Jersey, AD.676.972, Sept. 1968.

[28] S.R. KOSARAJU, "Limitations of DIJKSTRA's semaphore primitives and PETRI Nets", Research Report 25, John Hopkins University, Baltimore, May 1973.

[29] M. COURVOISIER, "Etude des Systèmes Logiques de Commandes Asynchrones à Evolutions Simultanées", Thèse d'Etat, Univ. Paul Sabatier, Toulouse, Février 1974.

[30] T. AGERWALA, M. FLYNN, "Comments on Capabilities, Limitations and Correctness of PETRI Nets", First Annual Symp. on Computer Architecture, Florida, 1973, pp. 81-86.

[31] F.G. HEATH, D. BAIN, "Simplified method for establishing determinacy in a directed-graph control structure", Electronics Letters, Vol. 11, N°1, 1975.

[32] C. RAMCHANDANI, "Analysis of asynchronous concurrent systems by PETRI net, Ph.D., M.I.T., 1973.

[33] M. HACK, "Decision problems for PETRI Nets and Vector Addition Systems", MAC Technical memorandum 59, M.I.T., March 1975.

[34] K. LAUTENBACH, H.A. SCHMID, "Use of PETRI Nets for proving correctness of concurrent process systems", Information Processing 74, North Holland Publishing Company.

[35] R. VALETTE, R. PRAJOUX, "A Model for Parallel Control Systems and Communication Systems", Conf. on Inf. Sciences and Systems, The Johns Hopkins University, Maryland, March 31/April 1,2, 1976.

[36] J.D. NOE, G.J. NUTT, "Macro E-Nets for Representation of Parallel Systems", IEEE Trans. on Computers, C-22, N°8, August 1973, pp. 718-727.

[37] K.E. IVERSON, "A Programming Language", New York, J. Willey, 1972.

[38] M. MOALLA, J. SIFAKIS, M. ZACHARIADES, "Un langage d'aide à la conception et à la simulation de systèmes complexes", Rapport de Recherche ENSIMAG, RR N°16, Oct. 1975.

[39] M. MOALLA, G. SAUCIER, J. SIFAKIS, M. ZACHARIADES, "A Design Tool for the Multilevel Description and Simulation of Systems of Interconnected modules", 3rd Annual Symp. on Comp. Architecture, Tanapa, Fla, Jan. 1976.

[40] P. LE DANOIS, M. MOALLA, G. SAUCIER, J. SIFAKIS, M. ZACHARIADES, "Multilevel Description and Simulation of Parallel Cooperating Processors",Kolloquium über das Parallelismus in der Informatik, Erlangen, RFA, Juin 1976.

[41] M. MOALLA, J. SIFAKIS, M. ZACHARIADES, "M.A.S., Un outil d'Aide à la Description et à la Conception des Automatismes Logiques", Colloque AFCET : Automatismes Logiques, Recherches et Applications Industrielles, Paris 6-8 Déc. 1976.

[42] R.G. GOUNTANIS, N.L. VISS, "A Method of Processor Selection for Interrupt Handling in a Multiprocessor System", Proc. IEEE, Vol. 44, N°42, Dec. 1966, pp. 1812-1819.

[43] R. VALETTE, "Sur la Description, l'Analyse et la Validation des Systèmes de Commande Parallèles", Thèse d'Etat, Univ. Paul Sabatier, Toulouse, Nov. 1976, p. 32.

[44] F. SCHUMACHER, "Simulation of Transport and Storage of Spent Nuclear Fuel", Proc. of the Int. Symp. and Course SIMULATION'75, Zurich, June 1975, pp. 408-413.

[45] M. ZACHARIADES, "Langage d'aide à la conception des systèmes logiques", Rapport DEA, ENSIMAG, Thèse 3ème Cycle à paraître.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU les rapports de présentation de :

Madame SAUCIER, Maître de Conférences à l'E.N.SI.M.A.G.
Monsieur LEBRUN, Chef du Service Diagnostics à la CII-HB
78 LES CLAYES-sous-BOIS

Monsieur Mohamed MOALLA

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "GENIE INFORMATIQUE"

A Grenoble, le 10 Décembre 1976,

Ph. TRAYNARD
Président
de l'Institut National Polytechnique