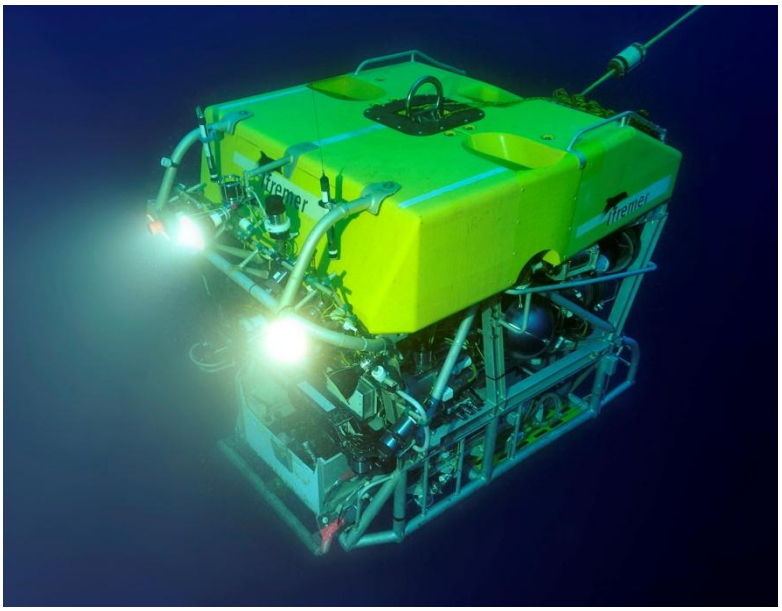
# Conception et développement du pilotage de 7 variateurs par Bus CAN pour le Système Temps-Réel du Victor 6000

Luc Tailliez

Centre IFREMER Méditerranée
Zone Portuaire de Brégaillon
BP 330
83507 La Seyne sur Mer Cedex

Luc Tailliez

# Conception et développement du pilotage de 7 variateurs par Bus CAN pour le Système Temps-Réel du Victor 6000




***Rapport de stage de 3ème année***
*Spécialité : Télécoms*

**Tuteur entreprise :** Mr Patrick Jaussaud

**Tuteur enseignant** : Mme Nadège Thirion-Moreau

**Année Universitaire 2014-2015**

# Engagement de non plagiat.

Je soussigné, Tailliez Luc

N° carte d'étudiant :

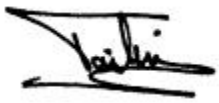
Déclare avoir pris connaissance de la charte des examens et notamment du paragraphe spécifique au plagiat.

Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document publiés sous quelques formes que ce soit (ouvrages, publications, rapports d'étudiant, internet etc…) constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour produire et écrire ce document.

Fait le 21/07/15 à Toulon

Signature(s)



Ce document doit être inséré en première page de tous les rapports, dossiers et/ou mémoires.

*Document du chapitre 10 annexe 5, issu de la Charte des examens adoptée en Conseil d'Administration le 11 juillet 2013 après avis du CEVU du 27 juin 2013 - Délibération N°2013-73*

# 1. REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage.

Tout d'abord, je tiens à remercier vivement mon maitre de stage, Mr Patrick Jaussaud, responsable des systèmes temps-réels au sein du service Systèmes Electriques et Electroniques Embarqués de l'Ifremer, pour son accueil, le temps passé ensemble et le partage de son expertise au quotidien.

Ma tutrice, Mme Nadège Thirion-Moreau, pour tous les cours de Programmation Orienté Objet, point-clé de ce stage, ainsi que les cours de traitement du signal qui m'ont passionné durant ces deux années universitaires.

Je remercie également toute l'équipe du service Systèmes Electriques et Electronique Embarqués pour leur accueil, leur esprit d'équipe et en particulier Mr Bruno Galizi qui m'a beaucoup apporté sur la partie électrotechnique du projet.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage.

## 2. SUJET

**Sujet de stage 3A SEATECH**

Spécialité Télécoms

# Conception et développement du pilotage de 7 variateurs par Bus CAN pour le Système Temps-Réel du Victor 6000

Dans le cadre de l'évolution des fonctionnalités du robot Victor 6000, l'Ifremer souhaite pouvoir interfacer au système temps-réel des équipements communiquant sur Bus CAN. Une application envisagée à court terme est le remplacement des variateurs de propulsion.

Le système temps-réel devra piloter sept variateurs par l'intermédiaire du Bus CAN. Il sera chargé de la configuration et la gestion du dialogue CAN et assurera le contrôle-commande des variateurs.

Ce projet sera réalisé au sein du Service Systèmes Electriques et Electronique Embarqués de l'Ifremer dont le rôle est entre autres la maintenance et les évolutions des systèmes temps-réel des engins sous-marins.

Le travail sera effectué en majeure partie en plateforme de simulation informatique. Les outils mis à dispositions se composent de machines de développement permettant de mener à bien la réalisation du projet, et de machines de simulation pouvant reproduire l'intégralité du fonctionnement de l'engin sous-marin Victor 6000.

Pour parvenir à l'aboutissement du projet et sa réussite, les phases suivantes devront être abordées chronologiquement :

- Recette du composant générique CAN (développé par Systerel)
- Développement d'un composant de pilotage des variateurs (ACE - C++)
- Intégration des composants aux feuilles de configuration (ACE – Protel)
- Mises à jour de l'interface graphique (Photon MicroGUI)
- Essais en local informatique de simulation
- Essais sur banc de tests avec variateurs et moteurs

Enfin, une fois testée et approuvée, cette nouvelle architecture de dialogue et de pilotage des variateurs sera prévue d'être mise en place sur l'engin lors du grand carénage en 2016.

# SOMMAIRE

# 3. ABREVIATIONS

| | |
|---|---|
| ACE | Automated Control Engine |
| API | Application Programming Interface |
| CAN | Controller Area Network |
| CEM | Compatibilité Electromagnétique |
| CSMA/CD | Carrier Sense Multiple Access with Collision Detection |
| D-List | Diagnostic List |
| E/S | Entrées / Sorties |
| GUI | Graphical User Interface |
| MLI | Modulation à largeur d'impulsion |
| OSI | Open Systems Interconnection |
| ROV | Remotely Operating Vehicle |
| SIS | Système Informatique de Surface |
| STR | Système Temps-Réel |
| TT | Télétransmission |

# 4. BIBLIOGRAPHIE


[1] **QNX**
https://en.wikipedia.org/wiki/QNX

[2] **Documentations ISE**
(doc. ACE, Protel, Photon microGUI, etc…)

[3] **Spécification technique d'un variateur de vitesse générique de propulseurs électriques** (doc. Ifremer DOP/DCM/SM/S3E/11-066)

[4] **Variateur de vitesse 3 phases pour moteur asynchrone**
(doc. Puissance+ MU-M03740-01.pdf)

[5] **CAN - Controller Area Network**
https://en.wikipedia.org/wiki/CAN_bus

[6] **Victor refonte variateurs : Spécifications du dialogue STR – Variateurs**
(doc. Ifremer IMN/SM/S3E/15-044)

[7] **Carte interface CAN-CPCI/331-2**
(doc. ESD : can-api_part1_function_manual_45)

[8] **Ifremer ROV Victor Driver CAN**
(doc. Systerel : C723_DCP_DRV_CAN)

[9] **Note – Augmentation du nombre de timers sous QNX4.25**
(doc. Note interne Ifremer P. Jaussaud)

# 5. PRESENTATION DE L'ENTREPRISE

L'Ifremer, Institut Français de Recherche pour l'Exploitation de la Mer, est un établissement public à caractère industriel et commercial (EPIC) placé sous la tutelle des ministères de l'Ecologie, de l'Energie, du Développement durable, de la Mer, de l'Alimentation et de l'Agriculture et de la Pêche.

Créés en 1984 par la fusion de deux organismes, l'ISTPM, Institut Scientifique et Technique des Pêches Maritimes, et le CNEXO, Centre National pour l'Exploitation des Océans, l'Ifremer a pour mission de connaître, évaluer et préserver les océans pour assurer leur durabilité.

Cet Institut, unique en son genre en Europe, est réparti sur 26 sites sur le littoral métropolitain et en DOM-TOM.

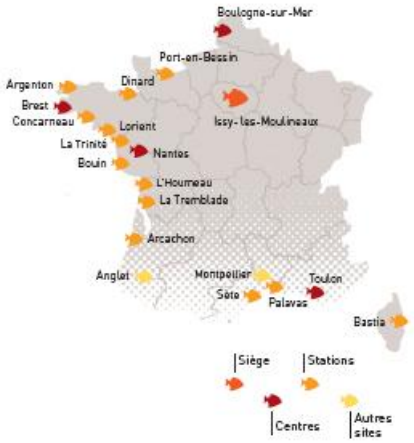



*Fig. 1 – Implantations Ifremer en métropole*

Avec un budget annuel de 214 millions d'euros hors opérations internes et 1528 salariés répartis dans les 5 centres opérationnels couvrant l'ensemble des façades maritimes françaises (Manche – Mer du Nord, Bretagne, Atlantique, Méditerranée et Pacifique), l'Ifremer contribue par ses travaux à la connaissance et à la préservation des océans et de leurs ressources. Il contribue également au développement, à la gestion et la mise à disposition de grandes infrastructures de recherche telles que la flotte, les moyens de calculs et les structures expérimentales, de la communauté scientifique nationale et européenne.

## 5.1. Le Centre Européen de Technologies Sous- Marines (CETSM)

Le Centre Européen de Technologies Sous-Marines est implanté sur le Centre Méditerranée de l'Ifremer dans la zone portuaire de Brégaillon à La Seyne-sur-Mer dans le Var. Cette localisation stratégique a été choisie pour des raisons géographiques et historiques.

Géographiques car la proximité de fonds sous-marins importants en fait un lieu de choix pour tester de nouveaux équipements sous-marins, et historique car l'implantation de la Marine Nationale permet de développer des partenariats et sous-traitances riches.

*Fig. 2 – Le centre IFREMER de la Seyne sur mer*

L'Ifremer Méditerranée accueille sur sa base marine l'armateur Genavir et des organismes scientifiques partenaires tels que l'Institut National des Sciences de l'Univers (INSU), le Centre de Documentation, de Recherche et d'Expérimentations sur les pollutions accidentelles des eaux (Cedre), l'Institut de Radioprotection et de sureté nucléaire (IRSN) ou le Centre d'Océanologie de Marseille (Université de Marseille – INSU).

## 5.2. L'Unité de recherche Systèmes Sous-Marins

A la Seyne-sur-Mer, l'Unité scientifique Systèmes Sous-Marins est chargée des développements et du suivi des systèmes et méthodes d'intervention, de reconnaissance et de surveillance sous-marine à caractère opérationnel ou exploratoire. Cette unité emploie une cinquantaine de personnes réparties au sein de trois services :

- Le Service Positionnement, Robotique, Acoustique et Optique (PRAO)
- Le Service Systèmes Electriques et Electronique Embarqués (S3E)
- Le Service Ingénierie d'Intervention et Développements Mécaniques (2IDM)

Les principales activités de cette unité sont :

- Le développement de nouveaux systèmes sous-marins en maitrise d'ouvrage tel que le HROV
- La conduite des évolutions des moyens opérationnels sous-marins
- La R&D sur les systèmes sous-marins opérationnels en relation avec le secteur industriel.
- La préparation et la conduite d'opérations d'intervention sous-marine spécifiques (épaves, recherche et autres affrètements).

- La réalisation d'expertises et de prestations utilisant ses moyens d'essais (piscine eau de mer, caisson d'essais hyperbare).

Les moyens sous-marins dont elle dispose :

- **Sous-marin Nautile** – construit en 1984 et capable de plonger à 6000m, il peut embarquer à son bord 3 personnes. Il comptabilise plus de 2000 plongées d'intervention sous-marines ou à caractère scientifique.

- **ROV Victor 6000** – robot télé-opéré depuis la surface et pouvant atteindre 6000m, il peut travailler 24h/24h. Il est notamment utilisé pour des missions de surveillance et de prélèvements d'échantillons pour la communauté scientifique.

- **AUV AsterX et IdefX** – véhicules sous-marins autonomes dédiés à la reconnaissance scientifique jusqu'à une profondeur de 2800 mètres. Ils peuvent embarquer un sondeur multifaisceaux, un sondeur de sédiment, un magnétomètre, un sondeur de pêche, un profileur de courant ou encore d'autres capteurs physiques.

- **Système Sismique Fond SYSIF** – constitué d'un lest sismique instrumenté, remorqué près du fond et mis en œuvre à partir d'un câble électro-optique, il remorque une flute sismique de réception afin d'enregistrer des coupes sismiques du plancher océanique jusqu'à 6000 m de fond.

- **HROV Ariane** – ROV Hybride principalement dédié à des applications côtières et jusqu'à 2500 mètres de fond. Il embarque à son bord sa propre énergie mais peut être aussi bien déployé en mode télé opéré (ROV) qu'en mode autonome (AUV).

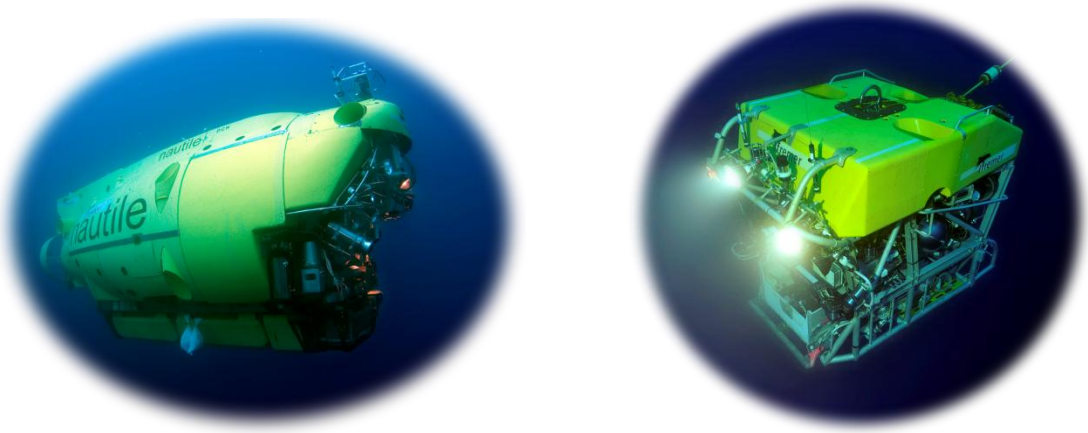


*Fig. 3 – Le sous-marin Nautile et le ROV Victor 6000 en plongée*

# 6. INTRODUCTION

## 6.1. ROV Victor 6000

Le Victor 6000 fait partie de la famille des ROV (Remotely Operated Vehicule), c'est un engin télé-opéré à câble qui a la particularité d'avoir un poids neutre dans l'eau grâce à son ballast.

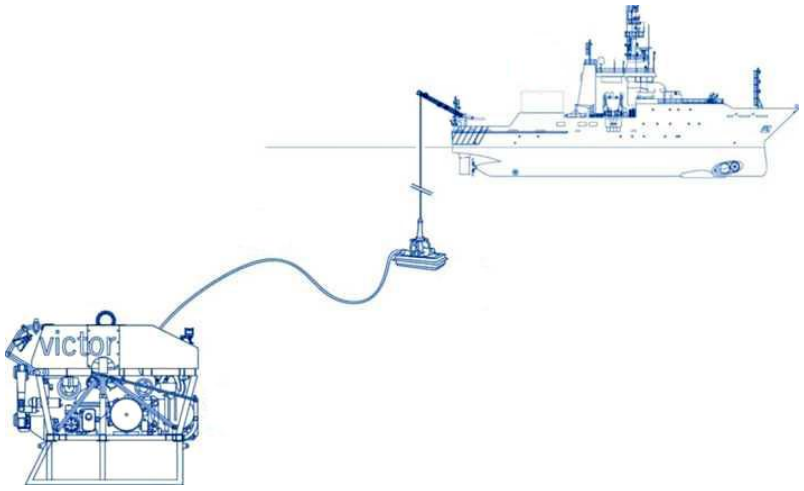



*Fig. 4 – Système Victor 6000*

Le câble, maintenu en tension par un lest, est déployé jusqu'à l'immersion souhaitée. Le ROV peut travailler dans un cercle autour du lest, défini par la longueur de l'ombilical souple, la laisse. Quand on veut changer de zone de travail, on déplace le bateau.

L'engin est alimenté à travers le câble et la laisse dans lesquels circulent les informations entre le fond et la surface par fibre optique.

Il est principalement destiné à des fins scientifiques et les équipes opérationnelles travaillent en régime de quart pour exploiter le ROV 24/24h.

## 6.2. Système temps-réel

Comme la plupart des engins asservis, le Victor intègre un système temps-réel (STR) qui permet la gestion et le pilotage de l'engin et des équipements. Il propose une interface graphique à dalle tactile à l'utilisateur pour faciliter le pilotage, le contrôle-commande des équipements et la gestion des modes automatiques et fait le lien vers le système informatique de surface (SIS).

Avant 2010, le STR était situé dans l'engin à l'intérieur d'un caisson étanche. Devenant obsolète, il a été remplacé et déporté en surface, libérant ainsi de la place sur l'engin pour de nouveaux équipements. Cela a été rendu possible grâce à l'amélioration du système de transmission par fibre optique.

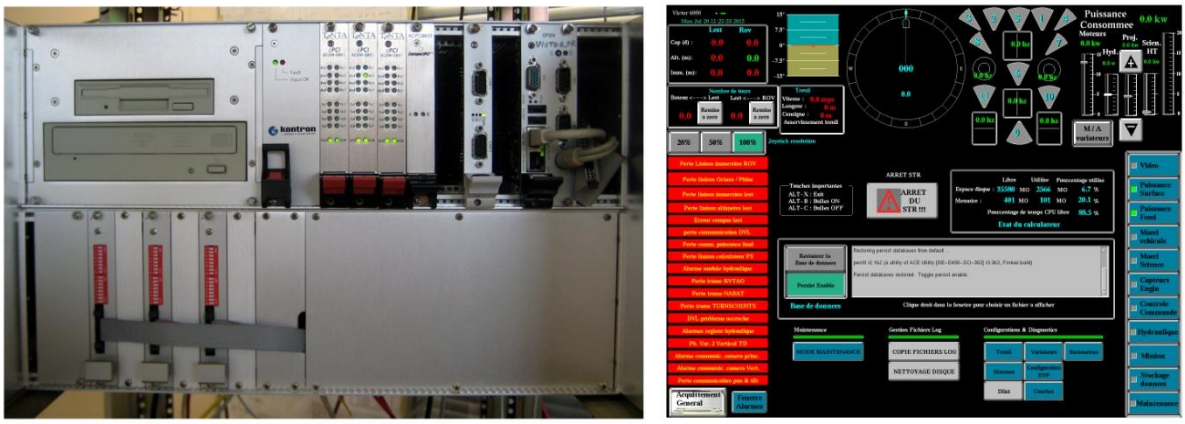


*Fig. 5 – Système temps-réel dans sa baie et interface graphique tactile du Victor 6000*

Son système d'exploitation est basé sur QNX 4.25, micro-noyau robuste, léger, rapide et complet, conçu principalement pour les systèmes embarqués. Il est compatible POSIX (standardisation des logiciels et interfaces de programmation pour UNIX). [1]

Le logiciel de contrôle, ACE est fourni par la société ISE, spécialisée dans les véhicules sous-marins. Il fournit des librairies de composants, encapsulé en classes C++, dont les fonctions sont exportées à travers l'interface de programmation (API). Les composants sont placés et connectés graphiquement par des signaux dans des feuilles de configuration (feuilles PROTEL) pour représenter le flux de contrôle des données. Cette disposition graphique des éléments minimise les erreurs et facilite le design et la compréhension. L'interface graphique (GUI) est réalisée sous Photon microGUI, console graphique de QNX. Il fournit aussi une liste de diagnostic (D-List) qui permet de visualiser et modifier tous les signaux déclarés dans le projet. [2]

## 6.3. Variateurs

Les variateurs actuels doivent être remplacés pour des raisons d'obsolescence de l'électronique.

Les nouveaux variateurs, développés spécialement pour Ifremer par Puissance+, peuvent alimenter tout moteur triphasé asynchrone fonctionnant en boucle ouverte, jusqu'à une puissance absorbée de 6kW. Ils sont donc compatibles avec tous les moteurs des engins du parc Ifremer. Appelé aussi convertisseur, ce variateur est composé de deux parties :

- Un onduleur triphasé à modulation de largeur d'impulsion (MLI) qui génère une tension triphasée ajustable sans neutre.
- Un circuit de commande qui pilote l'onduleur et contrôle en permanence son état.


*Fig. 6 – Variateur Puissance+ et sa carte interface*

Le dialogue s'effectue par liaison série et permet de configurer, connaître l'état et piloter le variateur. Une carte interface, procèdera à la relation entre le Bus CAN de l'engin et la liaison série du variateur, ainsi que son identification. Cette carte devra donc gérer un regroupement des données en entrée/sortie du Bus CAN d'une part et adresser les registres du variateur individuellement d'autre part. [3][4]

## 6.4. Réseau Bus CAN et implantation sur l'engin

### 6.4.1. Bus CAN

Le Bus CAN est un réseau à part entière respectant le modèle d'interconnexion des systèmes ouverts OSI (Open Systems Interconnection). C'est un réseau de terrain car il doit fonctionner dans un environnement limité et sévère comme une usine, une voiture, un sous-marin…

Le protocole CAN (Control Area Network) est un protocole de communication série qui supporte des systèmes temps réel avec un haut niveau de fiabilité. On retrouve ainsi dans le protocole CAN, la couche liaison de données (couche 2) et la couche physique (couche 1).

Il met en application une approche connue sous le nom de multiplexage, qui consiste à raccorder à un même câble (un bus) un grand nombre de calculateurs qui communiqueront donc à tour de rôle.

Chaque équipement connecté, appelé "nœud", peut communiquer avec tous les autres. Chaque nœud est connecté au bus par l'intermédiaire d'une paire torsadée (blindée ou non) et les deux extrémités du bus doivent être rebouclées par des résistances de 120Ω.

L'accès au Bus CAN suit la technique CSMA/CD (écoute de chaque station avant de parler mais pas de tour de parole, résolution des collisions par priorité).

Il est donc parfaitement adapté pour servir aux échanges de données dans les engins sous-marins. Il est déjà utilisé sur le Nautile (variateurs et projecteurs) ainsi que le HROV (variateurs et bras manipulateurs). [5]

### 6.4.2. Ancienne implantation liaison série RS422

Actuellement, l'architecture de dialogue pour les moteurs en place sur le Victor 6000 est composée d'une liaison série fond-surface RS422. Dans l'engin, une carte interface de multiplexage répartit les messages vers les 7 variateurs.

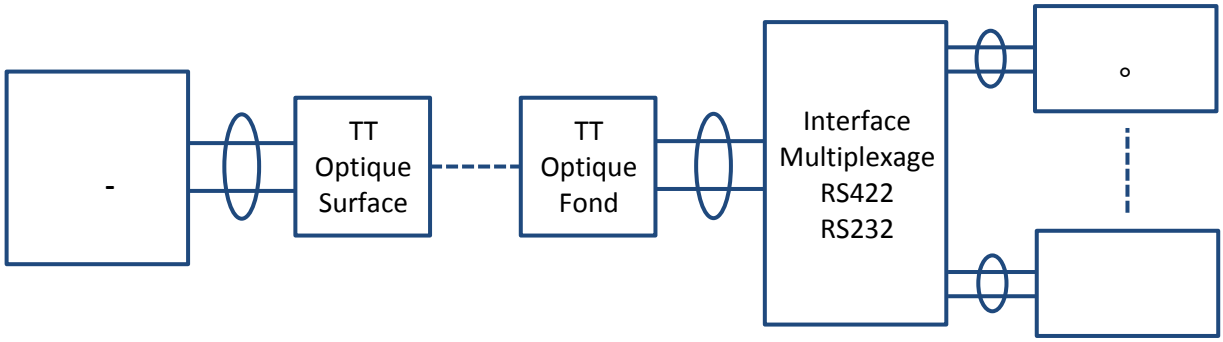



*Fig. 7 – Synoptique de dialogue des variateurs par liaison série*

### 6.4.3. Nouvelle implantation Bus CAN

Avec la nouvelle architecture de dialogue par Bus CAN, les éléments sont positionnés sur un bus de données et peuvent échanger les informations par son intermédiaire.

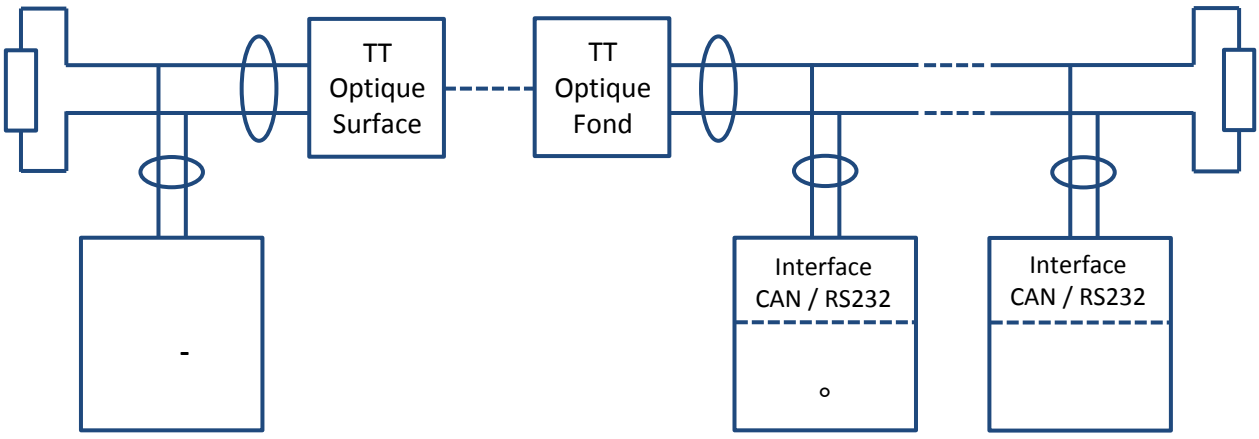



*Fig. 8 – Synoptique de dialogue des variateurs par Bus CAN*

Les télétransmissions (TT) optiques fond et surface ont été équipées de cartes Bus CAN, testées à 250kbits/s. La partie optique peut être considérée "transparente" dans la chaîne de transmission des données.

## 6.5. Synthèse des spécifications de dialogue Bus CAN

### 6.5.1. Contraintes de fonctionnement

- Le dialogue avec le STR doit permettre d'assurer la configuration et le dialogue des variateurs à partir de l'IHM.
- Le STR doit également prendre en compte la gestion des alarmes variateurs.
- La perte de dialogue doit être gérée de part et d'autre par des "timeout" et des actions associées.
- La période de rafraîchissement des consignes sera égale à 100ms, temps de cycle (Tc).

### 6.5.2. Principe de fonctionnement

Le calculateur temps-réel est le maître, le module variateur l'esclave. Le maître prend toujours l'initiative du dialogue et l'esclave lui répond. Le dialogue a deux modes de fonctionnement :

- Le mode "attente" où les variateurs sont en attente de n'importe quelle commande sans timing imposé, avec moteurs stoppés. Le STR peut dans ce mode faire la lecture/écriture de la configuration ou la lecture de l'état des variateurs.

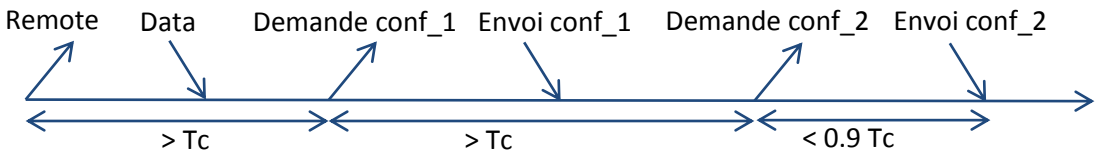



*Fig. 9 – Chronogramme des configurations*

- Le mode "normal" où les variateurs reçoivent les consignes à la période de rafraîchissement Tc et répondent par un état. Le module variateur bascule dans ce mode à la première réception d'une trame de consigne nulle et les défauts acquittés.

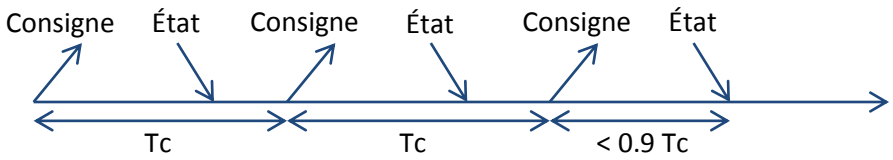



*Fig. 10 – Chronogramme des consignes*

Les défauts de transmissions (erreur de CRC, commande inconnue, valeurs hors plage, trame perdue) doivent être détectés et signalés. Au bout de 5 trames perdues consécutivement, le moteur doit être stoppé.

### 6.5.3. Particularité des trames

Les trames CAN seront standard au format CAN 2.0A. L'identifiant (ID) est codé sur 11 bits et la donnée est au maximum de 8 octets.

L'ID du message CAN est utilisé d'une part pour identifier le nœud destinataire ou émetteur du message (ID_VAR) et d'autre part pour préciser le type de message (ID_MSG). Seulement 8 bits suffisent et seront utilisés pour le codage de cette application :

| | N.U. | | | ID_VAR | | | | ID_MSG | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

*Fig. 11 – Répartition des bits de l'identifiant du message CAN*

L'ID_VAR permet l'affectation des variateurs de 1 à 7, respectivement : Transverse Avant, Vertical Tribord, Transverse Arrière, Longitudinal Tribord, Vertical Bâbord, Longitudinal Bâbord et la Centrale Hydraulique.

Le tableau ci-dessous liste les ID_MSG :

| ID_MSG | Type | Emetteur | Data Lenght | Description |
|---|---|---|---|---|
| 1 | Remote | STR | 0 | Demande configuration_1 |
| 1 | Data | VAR | 8 | Envoi configuration_1 |
| 2 | Remote | STR | 0 | Demande configuration_2 |
| 2 | Data | VAR | 8 | Envoi configuration_2 |
| 3 | Data | STR | 8 | Ecriture configuration_1 |
| 4 | Data | STR | 8 | Ecriture configuration_2 |
| 5 | Data | STR | 8 | Trame de consigne_1 |
| 6 | Remote | STR | 0 | Demande état_1 |
| 6 | Data | VAR | 8 | Envoi état_1 |
| 7 | Data | STR | 8 | Trame de consigne_2 |
| 8 | Remote | STR | 0 | Demande état_2 |
| 8 | Data | VAR | 8 | Envoi état_2 |

*Fig. 12 – Liste des messages CAN*

Il existe 2 trames de consigne dont seuls les identifiants changent, renvoyant des états différents. Afin de recevoir tous les états du variateur, le STR alternera avec une trame de consigne 1 et une trame de consigne 2. [6]

# 7. REALISATIONS STR / ACE

## 7.1. Matériel et composant CAN Générique

Côté matériel, le choix s'est porté sur une carte au format Compact PCI à ajouter au calculateur temps-réel existant. Le fabriquant ESD propose une carte avec deux interfaces CAN et disposant des drivers pour QNX 4.25. La référence de ce matériel est : ESD CAN-CPCI/331-2. [7]

Côté logiciel, l'Ifremer a demandé une prestation à la société Systerel pour réaliser un composant ACE générique permettant d'offrir les services rendus par le driver CAN au niveau de la programmation graphique. [8]

## 7.2. Banc de tests

Les réalisations des composants se faisant en local informatique, il est nécessaire de pouvoir simuler chaque variateur et tester le fonctionnement de ses variables de configuration et d'état, les consignes, les alarmes et le dialogue.

La carte ESD a deux interfaces CAN qui seront utilisées pour dialoguer entre les composants de pilotage, de simulation et les variateurs. Un terminal espion est inclus dans la chaîne pour recevoir et envoyer des messages CAN.

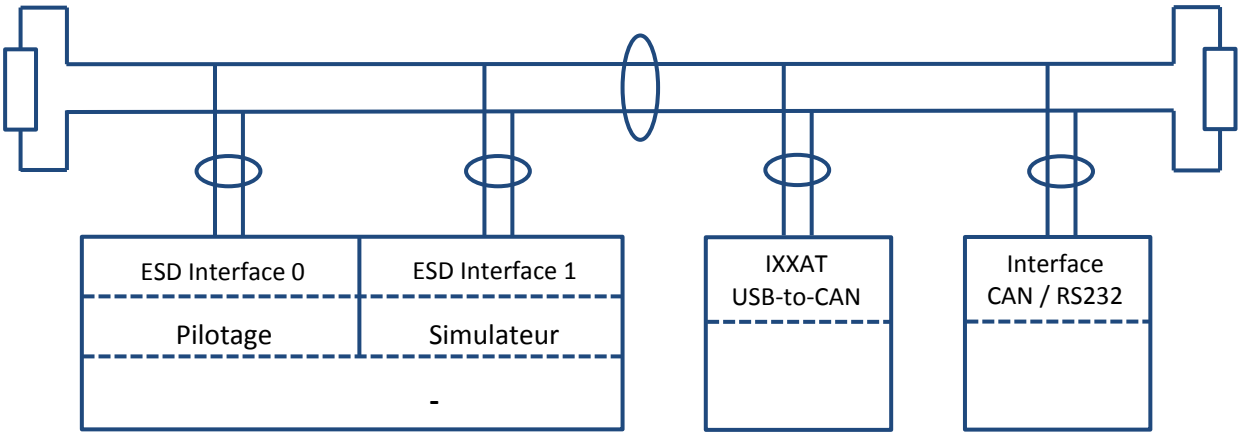



*Fig. 13 – Banc de tests des variateurs*

## 7.3. Choix de programmation pour le pilotage des variateurs

A partir de l'ancien composant de gestion des moteurs "victor_thrusters", qui pilotait tous les variateurs par liaison série, et du composant générique Bus CAN "victor_can_driver", il sera conçu un composant unique de gestion des moteurs qui sera utilisé indépendamment pour chaque variateur. Ce composant aura les avantages suivant :

- Toutes les variables du variateur Puissance+ sont présentes sur ses entrées / sorties (E/S)
- Le driver CAN est directement intégré (simplifie les échanges vers le Bus CAN)
- Indépendance inter-variateurs (pilotage, dialogue, isolation)

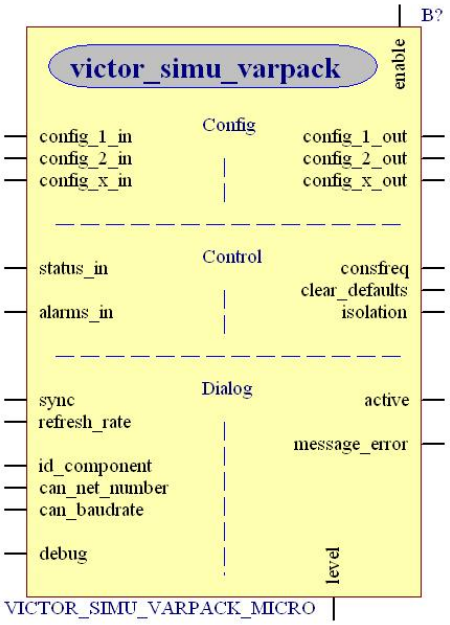
Le composant sera partagé en trois blocs distincts :

- Config – E/S de configuration
- Control – E/S de consignes, d'états et d'alarmes
- Dialog – Gestion des actions du composant, configuration et états du dialogue CAN

## 7.4. Composant de simulateur de variateur

La première classe C++ réalisée, "VictorSimuVarpack", est celle du simulateur qui va permettre d'obtenir une classe proche de celle du composant de pilotage, par ses E/S presque similaires et vérifiant le portage des méthodes liées au Bus CAN, récupérées dans la classe du composant "victor_can_driver" de Systerel.

Ce composant ne servant qu'à émuler les trames du variateur, sa partie dialogue est minimalisée pour la configuration du CAN et le taux de rafraîchissement.




| Classe "VictorSimuVarpack" | |
|---|---|
| Méthodes publiques | |
| Standard | define, install, initialize |
| Action | enable |
| Action | SyncProc |
| Action | DebugLevelProc |
| Méthodes privées | |
| Subscribe, Unsubscribe | |
| ReadMessage, SendMessage | |
| GetMessageData | |
| PrepareResponse | |
| DebugMsg, ShowMessage | |

*Fig. 14 – Composant de simulateur de variateur (vue réduite)*

Les signaux des entrées seront à renseigner manuellement par la D-List. Ils représentent les valeurs des variables de configuration, d'état et les alarmes du variateur pour constituer les messages de réponse aux requêtes. Les valeurs des sorties sont celles reçues par le variateur lors des envois de messages de configurations et de consignes.

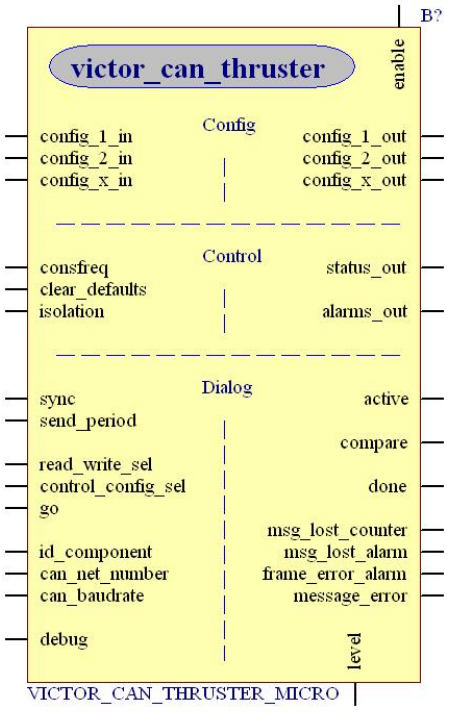
Son dialogue est configuré pour fonctionner sur l'interface n°1 de la carte ESD, à 250kbits/s. Chaque simulateur aura un identifiant de 1 à 7, correspondant au numéro du variateur.

Enfin, le composant doit être synchronisé par une horloge externe à 100Hz, soit une période de 10ms. Le taux de rafraichissement est un multiplicateur de la synchro. Pour un taux de 5, le message de réponse sera renvoyé au bout de 50ms. En augmentant ce taux, il sera possible de générer des "timeout" au niveau du composant de pilotage et donc de tester les alarmes de dialogue.

## 7.5. Composant de pilotage de variateur

### 7.5.1. Schéma du composant

La deuxième classe C++, "VictorCanThruster", est un portage de la classe du composant de simulation, dont la partie dialogue est complétée par la présence d'une double machine à états et sa gestion, les alarmes et informations qui en découlent.




| Classe "VictorCanThruster" | |
|---|---|
| Méthodes publiques | |
| Standard | define, install, initialize |
| Action | enable |
| Action | SyncProc |
| Action | DebugLevelProc |
| Méthodes privées | |
| Subscribe, Unsubscribe | |
| ReadMessage, SendMessage | |
| GetMessage, SetMessage | |
| CheckMessageType | |
| DebugMsg, ShowMessage | |

*Fig. 15 – Composant de pilotage de variateur (vue réduite)*

Son dialogue est configuré pour fonctionner sur l'interface n°0 de la carte ESD, à 250kbits/s. Chaque composant de pilotage aura un identifiant de 1 à 7, correspondant au numéro du variateur.

### 7.5.2. Synchronisation

Le composant doit être synchronisé par une horloge externe (sync) qui activera la méthode "SyncProc". Ce principe de cadencement permet au composant de ne pas contenir de boucle d'attente et de rendre la main dès l'action terminée.

La période d'envoi des messages sur le CAN (send_period) est un multiplicateur de la synchro et le "timeout" est calculé par la relation : timeout = 0.9 x send_period. Avec une horloge d'une période de 10ms et une période d'envoi égale à 10, on obtient une transmission des messages CAN à 100ms et un "timeout" à 90ms. Deux drapeaux de synchro, "T-out Flag" et "Send Flag", sont générés en début de méthode et seront utilisés par les machines à états.

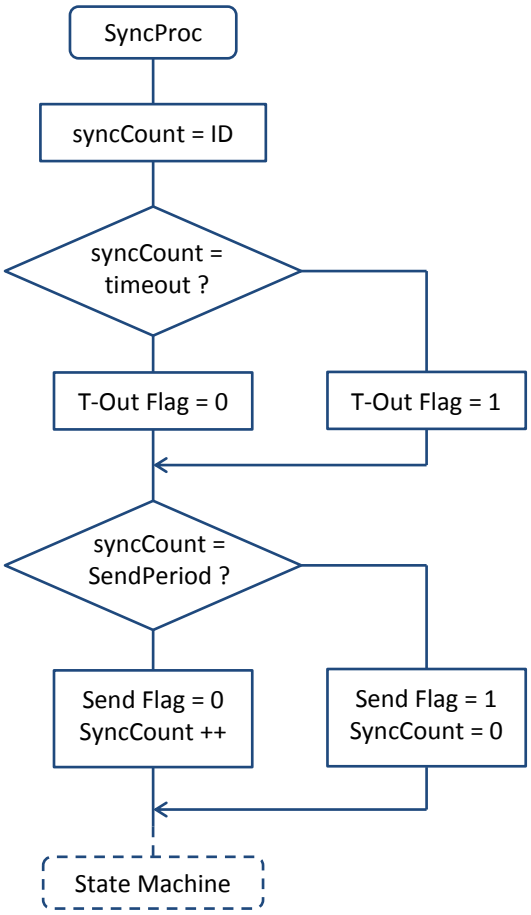



*Fig.16 – Organigramme de synchronisation*

Remarque : le compteur de synchro, "syncCount", est initialisé avec le numéro du variateur (id_component). Comme l'activation des composants (enable) est commune, les émissions de messages seront temporellement réparties afin d'étaler la charge de Bus CAN et le composant ayant l'identifiant le plus grand émettra le premier.

### 7.5.3. Machines à états

La première machine est destinée à la gestion des états et des actions du composant. Elle est capable de faire appel à la deuxième machine, dédiée à l'écriture et la réception des messages CAN,

en positionnant le drapeau "wrFlag". Les tableaux suivants sont volontairement en anglais facilitant ainsi le suivi avec le code C++ du composant de pilotage.

**wrFlag = 0**

| State | Event or action (A) | Result | Next State | Comments |
|---|---|---|---|---|
| INIT | Subscribe to CAN (A) | OK | ENABLE | Bus CAN init. |
| | | Not OK | ERROR | Bus CAN error |
| ENABLE | control_config_sel = 0<br>go = 1 | | SETPOINTS | Setpoints selection |
| | control_config_sel = 0<br>read_write_sel = 0<br>go = 1 | | READ_CONFIG | Read config selection |
| | control_config_sel = 0<br>read_write_sel = 1<br>go = 1 | | WRITE_CONFIG | Write config selection |
| | go = 0 | | ENABLE<br>WR_WRITE | State Machines init. |
| SETPOINTS | go = 1<br>wrFlag = 1 (A) | | SETPOINTS | Toggle setpoints frames |
| | go = 0 | | ENABLE | Exit |
| READ_CONFIG | wrFlag = 1 (A) | T-Out | TIMEOUT_ERROR | No timeout allowed while reading |
| | | No T-Out | READ_END | |
| READ_END | go = 1 | | READ_END | |
| | go = 0 | | ENABLE | Exit |
| WRITE_CONFIG | Send Config Frame (A) | OK | READ_CONFIG | Read to check writing |
| | | Not OK | ERROR | Bus CAN error |
| TIMEOUT_ERROR | Send T-Out msg. (A) | | ERROR_NO_MSG | Debug message |
| ERROR | Send error msg (A) | | ERROR_NO_MSG | Debug message |
| ERROR_NO_MSG | | | ERROR_NO_MSG | Init. needed |

**wrFlag = 1**

| State | Event or action (A) | Result | Next State | Comments |
|---|---|---|---|---|
| WR_WRITE | Send Frame (A)<br>@ Send Flag | OK | WR_READ | Synchronized writing |
| | | Not OK | WR_ERROR | Bus CAN error |
| WR_READ | T-Out Flag | Not occured | Immediate (2) | |
| | | Occured | Immediate (1) | Set msg_lost_counter |
| (1) | Max T-Out | Not occured | Immediate (2) | |
| | | Occured | WR_TIMEOUT | Set msg_lost_alarm |
| (2) | Read CAN msg. (A) | Received | WR_WRITE<br>wrFlag = 0 | Frame is received |
| | | Not received | WR_READ | Frame is not received |
| | | Not OK | WR_ERROR | Bus CAN error |
| WR_TIMEOUT | go = 1 | | WR_TIMEOUT | |
| | go = 0 | | ENABLE<br>wrFlag = 0 | Exit and init. State machine |
| WR_ERROR | | | ERROR<br>wrFlag = 0 | Bus CAN error |

*Fig.17 –fonctionnement des machines à états*

### 7.5.4. Programmation

Les classes C++ ont été ajoutées à la bibliothèque Victor 6000 de ACE pour compilation. Par gain de temps à la compilation, il a été judicieux d'inclure uniquement les classes des composants de pilotage et de simulation, de les tester jusqu'à arriver à des versions quasi-finales de ces classes.

L'intégralité des classes des composants a pu être finalement recompilée. Seules quelques modifications mineures sur les composants de pilotage et de simulation auront nécessité d'autres interventions.

## 7.6. Interfaçage des composants

Dans la mesure du possible, les signaux des E/S des nouveaux composants ont été connectés aux anciens dans les feuilles de configuration afin de conserver les anciennes dénominations, et réduire au maximum les modifications apportés aux feuilles PROTEL : réutilisation d'anciennes alarmes, de signaux similaires, des consignes de pilotage, etc...

Malgré tout, plus de 5000 signaux et de nombreux composants dans environ 30 feuilles de configuration ont été modifiés, renommés ou ajoutés !

Les notes du travail accompli sont disponibles en annexe et donnent un suivi des modifications.

Un travail minutieux avec des essais réguliers aura permis cette réalisation sans la moindre négligence.

### 7.6.1. Problème de Timers

Lors de la recette des composants CAN générique développés par Systerel, il a été décelé un problème de Timers. L'erreur n°22 (EINVAL) apparaissait lors de la souscription des composants sur le Bus CAN. Chaque souscription au Bus CAN nécessite un processus de gestion (handle) qui utilise 2 Timers, soit 28 Timers au total requis pour le dialogue Bus CAN des variateurs et les simulateurs.

Il a été nécessaire d'augmenter le nombre de Timers de 125 à 150. Cette opération nécessite de recompiler le noyau QNX4.25 et doit donc être réalisée avec prudence (sauvegardes, image de boot alternative à jour …). [9]

### 7.6.2. Variables persistantes

Il existe la possibilité d'avoir des variables persistantes, stockées dans des bases de données, qui seront restaurées à chaque nouvelle session.

Deux nouvelles bases de données ont été ajoutées :

- "**persist_can_thrusters.dat**" : pour le pilotage, sont présents les paramètres de configuration du Bus CAN et des composants, ainsi que les variables de configuration des variateurs

(propulseurs ou centrale hydraulique). Ces variables serviront lors de la lecture afin de vérifier l'intégrité des paramètres de configuration des variateurs, mais aussi pour l'écriture ou la mise à jour de nouveaux paramètres.

- "**persist_simu_varpacks.dat**" : pour les simulateurs, sont les paramètres de configuration du Bus CAN et des composants. Chaque simulateur aura ses propres variables persistantes de configuration qui auront pour rôle de simuler l'écriture et la lecture dans les registres du variateur.

Les fichiers originaux de ces données sont "persist_can_thrusters.txt" et "persist_simu_varpacks.txt".

### 7.6.3. Scripts

Pour gérer l'initialisation du composant de pilotage et l'écriture des paramètres des variateurs, il a été préférable de la faire sous la forme de deux scripts séquentiels qui allègent les feuilles de configuration :

- "**thruster_init.bsn**" : ce script d'initialisation active les composants de pilotage. Il est démarré soit par l'appui du bouton de procédure de démarrage "M/A Variateurs" de la page Puissance Fond, soit du bouton "Init" de la page d'interface des variateurs.

- "**thruster_write.bsn**" : ce script gère la procédure d'écriture des paramètres de configuration des variateurs. Il est activé par le bouton "Ecrire" de la fenêtre de confirmation associée à cette fonction.

# 8. INTERFACE GRAPHIQUE

Sous Photon microGUI, l'interface graphique est réalisée à l'aide de "Widgets" (composants graphiques) qui peuvent interagir avec les signaux des feuilles de configuration. Il faut d'une part déclarer le type de composant graphique utilisé et les évènements liés aux signaux (gui_file_xxx.ini), et d'autre part faire le lien avec le "Widget". Ainsi, il est aisé d'afficher ou de modifier la valeur de n'importe quel signal par ce biais.

## 8.1. Interface puissance fond

Rappels : **il est impératif de démarrer les variateurs par le bouton "M/A Variateurs"** car celui-ci permet la séquence complète d'initialisation.

Si les boutons "BT Variateurs" et "Variateurs 280V" sont activés dans un premier temps pour des raisons particulières, il faudra tout de même valider le bouton "M/A Variateurs" pour que le processus de démarrage soit achevé.

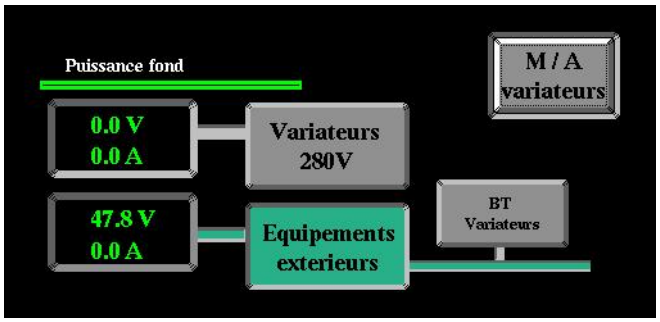



*Fig.183 – GUI – Puissance fond*

## 8.2. Interface des variateurs

Pour chaque variateur, on retrouve dans une boite, l'affichage des variables de contrôle les plus significatives (tension et courant du bus 280V, température des IGBT et les 3 courants phases), ses alarmes propres, l'état du son dialogue CAN et sa configuration. Il est toujours possible d'isoler un variateur en défaut pour problème d'isolement ou autre : **la tension du Bus 280V doit être coupée pour que le bouton d'isolement soit actif**.

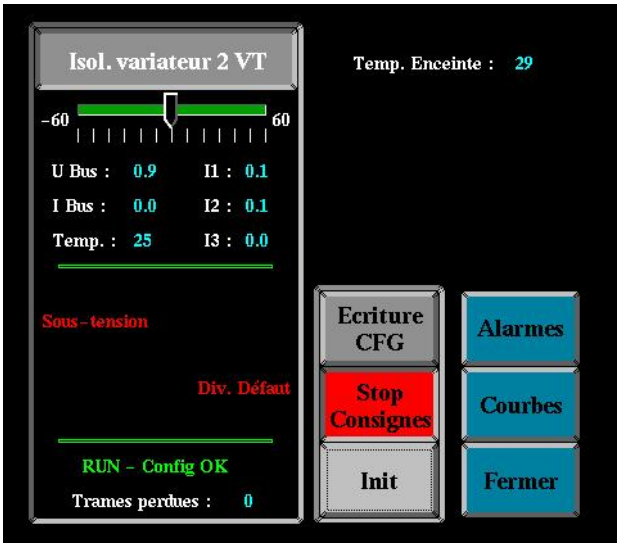



*Fig. 19 – GUI – Interface variateurs*

Il y a 5 états possibles pour un variateur :

- **RUN – Config OK** : émission de trames en cours, configuration vérifiée
- **RUN – Config Non OK** : émission de trames en cours, configuration erronée
- **STOP – Config OK** : pas d'émission de trames, configuration vérifiée
- **STOP – Config Non OK** : pas d'émission de trames, configuration erronée
- **Inactif** : le composant est désactivé

Les variateurs et leur carte interface sont préprogrammés pour répondre à un emplacement spécifique (propulseurs ou centrale hydraulique). Lors d'un remplacement de variateur, la

configuration doit être vérifiée et les valeurs des variables persistantes doivent correspondre avec les valeurs des variables du variateur : le message "**Config OK" apparaît en vert**. Dans le cas contraire, une vérification manuelle des variables dans la D-List s'impose car le variateur n'a certainement pas été monté au bon emplacement ou n'est pas bien paramétré.

Des boutons additionnels pour la maintenance proposent aussi d'initialiser le dialogue CAN (relance du script de démarrage), de stopper les consignes, de visualiser et définir les alarmes utilisées, d'afficher des courbes (vitesse, tension et courant) et d'écrire les configurations des variateurs : **l'opération d'écriture ne devra être réalisée que par un opérateur confirmé** car les valeurs des variables persistantes seront écrites et sauvegardées dans la mémoire des cartes variateurs. Pour effectuer l'écriture, les consignes doivent être stoppées et une fenêtre de confirmation demandera à l'utilisateur s'il est certain de vouloir effectuer la programmation.

La température de l'enceinte variateur est donnée par une sonde PT100 qui doit être câblée sur un des variateurs. Si plusieurs sondes sont présentes, la température de la sonde du variateur avec l'identifiant le plus grand sera lue. Si aucune sonde n'est connectée, la température sera de 99°C.

## 8.3. Alarmes des variateurs et leur signification

Les alarmes propres à chaque variateur sont contenues dans une boite. Il existe 10 alarmes spécifiques au variateur :

- 5 alarmes critiques : Haute Intensité, Haute Tension, Basse Tension, Haute Température, Défaut pack puissance provoquant l'arrêt du variateur et pour lesquelles les consignes seront stoppées.
- 2 alarmes non critiques : Courant instable et Rabattement de fréquence
- 1 regroupement d'alarmes diverses : Vbus > 400V, Vbus < 95V, Ibus > 90A, Uphnom > 0,7 x Udcmin, Defaut Alim BT, MiscDefaults (InfoDefAlimBT, InfoInhibHard, InfoDefHardIbus) consultables dans la D-List.
- 2 alarmes de dialogue : Erreur Bus CAN et Time-Out

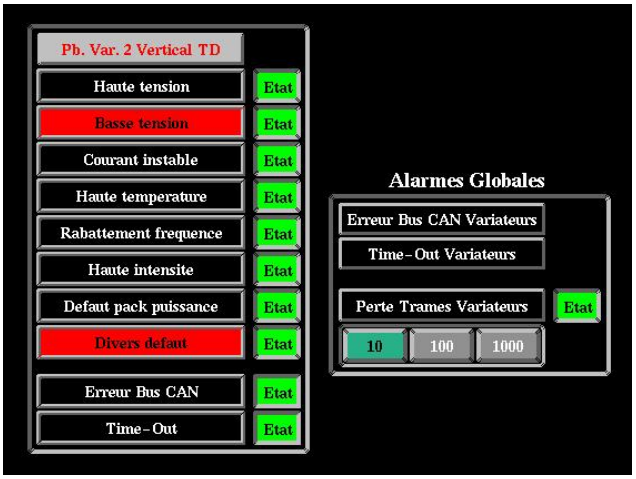



*Fig.20 – GUI – Interface variateurs*

Une boite d'alarmes globales est aussi disponible et inclut notamment l'alarme de Perte de trames variateurs qui peut être paramétrée pour un nombre de trames perdues.

Signification des alarmes et causes possibles :

| Alarmes | Signification | Causes possibles |
|---|---|---|
| Haute Tension | Surtension sur le bus 280V | Régulation du 2000V trop haute. Problème de délestage. |
| Basse Tension | Sous-tension sur le bus 280V | 280V Variateurs pas démarré ou fusible HS. Vérifier si la tension bus est correcte. |
| Courant Instable | Déséquilibre Courant phase | Possible sur le pont à faible courant (moteur sans charge). Défaut d'isolement furtif. |
| Haute Température | Temp IGBT > 100°C | Eau de mer trop chaude en surface. Diminuer les consignes. |
| Rabattement Fréquence | Limitation de courant | Huile centrale hydraulique froide. Diminuer les consignes propulseurs. |
| Haute Intensité | Surintensité courant phase | Vérifier les courants phases. Probable moteur bloqué ou défaut d'isolement. |
| Défaut Pack Puissance | Défaut | Variateur HS. IGBT HS. |
| Divers Défaut | Regroupement de plusieurs défauts | Consulter la D-List si nécessaire. |
| Erreur Bus CAN | Problème sur le CAN | Problème de câblage, de TT, carte CAN STR, driver, etc… |
| Time-Out | Max. de Time-Out successifs atteint | Variateur non connecté. Carte interface HS. |
| Pertes Trames Variateurs | Limite de trames perdues | Vérifier si un variateur perd des trames en particulier. Problème de carte interface. |

*Fig.21 – Liste des alarmes*

Il existe un cas exceptionnel où toutes les alarmes peuvent être activées en même temps et les configurations lues à 0 : la carte interface est bien présente mais le dialogue RS232 avec le variateur ne s'effectue pas ou mal. Un des drivers RS232 peut être HS ou il existe un problème de mode commun lors des commutations des IGBT.

# 9. ESSAIS

Les essais de dialogue en local simulation étant probants, il était nécessaire de compléter les tests avec un réel pilotage de moteurs.

## 9.1. Essais sur bancs de tests

Ces premiers essais ont été réalisés au local électrotechnique. Le schéma du banc de tests des variateurs (Fig. 13) a été reproduit en labo avec 2 ensembles variateurs et moteurs, et 5 simulateurs.

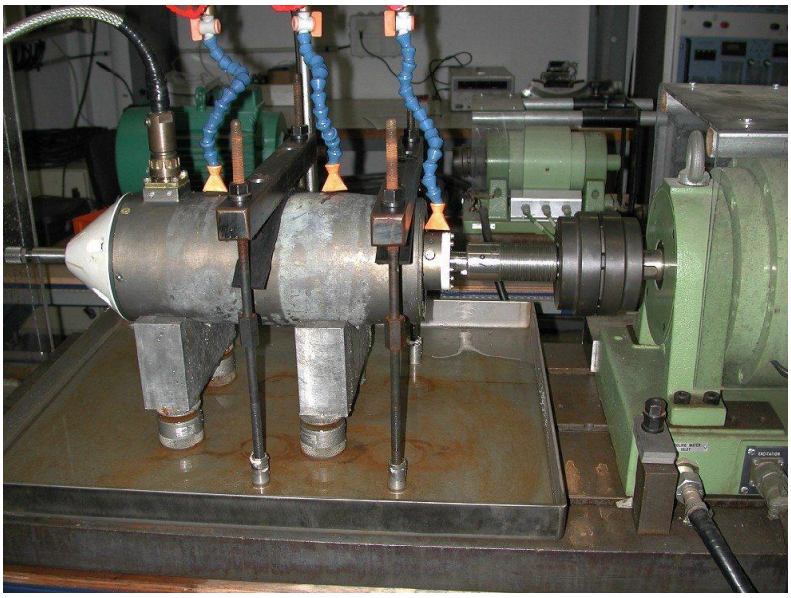


*Fig. 22 – Bancs de tests électrotechniques*

Il est apparu des problèmes de dialogue dès le démarrage des moteurs : sauts de valeurs, alarmes furtives et perte de trames sur le Bus CAN, jusqu'au "timeout" des variateurs.

Des modifications matérielles et logicielles ont dû être apportées à la carte interface afin de reprendre les essais. Une seule carte interface ayant été modifiée, les tests ont continué avec un seul ensemble variateur et moteur et les simulateurs.

Enfin, la majorité des alarmes et des états ont pu être testées ainsi que le pilotage des moteurs par le système temps-réel. Ces essais ont pu confirmer l'utilité des simulateurs et leur fiabilité à reproduire le fonctionnement des variateurs en local informatique.

## 9.2. Essais longue durée piscine

Afin de finaliser la recette des variateurs, des essais de longue durée ont été réalisés en piscine pour reproduire le fonctionnement normal des propulseurs.

*Fig. 23 – Moteur et tuyère montés dans une cage*

Cela aura permis de tester le dialogue avec le STR durant une longue période mais surtout aux électrotechniciens de mesurer la réponse du variateur, qui doit répondre aux exigences du cahier des charges imposé par Ifremer.

## 9.3. Problèmes de carte interface

Les problèmes de cette carte peuvent être abordés sous trois aspects différents :

- Logiciel : implanté sur microcontrôleur, il ne présentait pas de gestion d'erreurs (informations, traitements, recouvrements). Le logiciel a été entièrement revu et a permis de traiter les problèmes.

- Matériel : la carte doit opérer en milieu hostile : variations de température, bruit. Un nombre important de rectifications sur les composants ont été apportées pour satisfaire de meilleurs résultats face à ces agressions.

- Compatibilité Electromagnétique (CEM) : la carte est soumise à de fortes perturbations lors des commutations des IGBT (1000V, 1µs). Un plan de masse a été ajouté sous la carte réduisant à néants les effets de mode commun.

Des améliorations du blindage sont en cours d'étude et une révision complète de l'électronique serait souhaitable.

# 10. CONCLUSION

Ce projet de fin d'étude aura permis de me réintroduire très rapidement à la vie d'entreprise dont je m'étais détaché pendant deux ans pour suivre la formation Ingénieur Télécoms de SeaTech.

La partie informatique temps-réel, dont j'avais en charge la réalisation, a pu être démarrée dès les premiers jours avec un minimum de formation, ce qui m'a laissé le temps d'approfondir et de soigner encore plus particulièrement mon travail et de participer à l'intégralité du projet en y apportant mon savoir-faire.

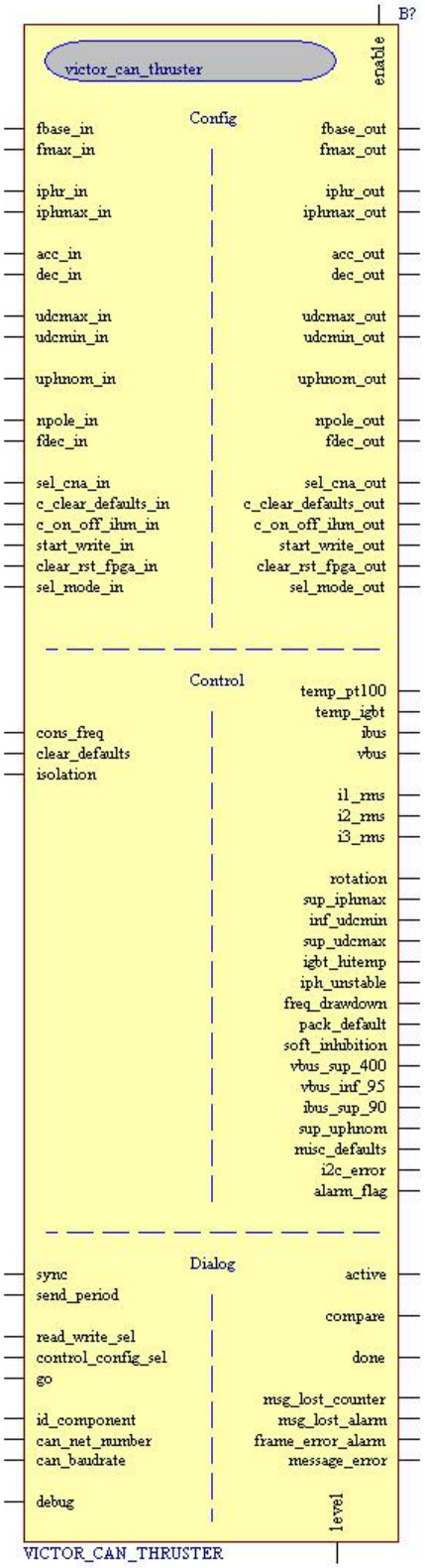
Le prochain arrêt technique du ROV Victor 6000 est prévu le 2ème trimestre 2016. D'ici là, de nouvelles cartes interfaces devraient être testées à partir de septembre et les problèmes de CEM résolus au plus vite. Il faudra ensuite attendre l'intégration des variateurs dans leur caisson étanche pendant l'arrêt technique. Pour qualifier ces nouveaux variateurs, de nombreux essais seront réalisés en piscine avec plusieurs moteurs, suivis par des plongées d'essais en darse pour enfin terminer par des plongées au large de Toulon pour homologuer définitivement les modifications.

Le travail accompli pendant ce stage a pu démontrer la mise en œuvre de mes compétences de futur ingénieur, étoffées par la formation Télécoms : d'une part en ce qui concerne l'organisation du temps de travail et la maîtrise des domaines de l'électronique et l'informatique, et d'autre part avec la transmission de mes connaissances et mon savoir technique lors du travail en équipe pendant les essais.

# 11. ANNEXES

## 11.1. Composant pilotage de variateur

### 11.1.1. Implémentation Protel



### 11.1.2. Liste des services

| Classe "VictorCanThruster" | | |
|---|---|---|
| Méthodes publiques | | Rôle |
| Standard | Define | Définition du composant et des attributs |
| Standard | install | Installation du composant et des attributs |
| Standard | initialize | Initialisation du composant et des attributs |
| Action | enable | Activation du composant et des attributs |
| Action | SyncProc | Cadencée par une horloge, 2 machines à états et une procédure de synchronisation vont gérer les différents modes d'utilisation du composant, le cadencement des trames sur le Bus CAN et la gestion des erreurs |
| Action | DebugLevelProc | Définition du mode de Debug utilisé |
| Méthodes privées | | Rôle |
| Subscribe | | Le calculateur inscrit un composant CAN au driver. Cela va lui permettre d'émettre/recevoir des messages vers/depuis ce composant |
| Unsubscribe | | Le calculateur désinscrit un composant CAN du driver |
| ReadMessage | | Vérifie si un message CAN a été reçu depuis un composant CAN défini sur le driver. Si un message a bien été reçu, transmet les données correspondantes en sortie du driver |
| SendMessage | | Envoie un message CAN vers un composant CAN défini sur le driver |
| GetMessage | | Dé-concaténation des données du message CAN vers les signaux de sorties |
| SetMessage | | Dé-concaténation des données du message CAN vers les signaux de sorties |
| CheckMessageType | | Vérification de l'entête du message |
| DebugMsg | | Automatisation du message de Debug |
| ShowMessage | | Affiche le message sur la sortie standard |

### 11.1.3. Liste des entrées / sorties

| Nom de l'attribut | Type de l'attribut | Flot de Données | Gamme | Instanciation | Description |
|---|---|---|---|---|---|
| Paramètres standard | | | | | |
| level | %level | - | - | obligatoire | Niveau de priorité |
| enable | %integer | entrée | actif (≠0)<br>inactif (0) | optionnelle | Active ou désactive le composant. Si non déclarée, le composant est toujours activé |
| debug | %integer | entrée | 0 à 4 | optionnelle | Mode de Debug Utilisé. Si non déclaré, pas de Debug |
| Dialogue - Synchronisation et paramètres | | | | | |
| sync | %sync | entrée | - | obligatoire | Synchronisation du composant par une horloge à 100Hz |
| send_period | %integer | entrée | 100 à 500 | obligatoire | Période d'envoi des trames sur le CAN. Le timeout est calculé par timeout = 0,9 x send_period |
| read_write_sel | %integer | entrée | écriture (≠0)<br>lecture (0) | obligatoire | Sélectionne la lecture ou l'écriture des configurations quand control_config_sel = 1 |
| control_config_sel | %integer | entrée | config (≠0)<br>control (0) | obligatoire | Sélectionne les modes envoi de consignes ou configuration du variateur |
| go | %integer | entrée | démarre (≠0)<br>stop (0) | obligatoire | Exécution de l'action sélectionnée à la prochaine synchronisation de sync. La sélection des actions est réalisée par read_write_sel et control_config_sel |
| id_component | %integer | entrée | 1 à 7 | obligatoire | Numéro d'identification du composant |
| can_net_number | %integer | entrée | 0, 1 | obligatoire | Numéro de l'interface CAN de la carte ESD à utiliser pour le dialogue |
| can_baudrate | %integer | entrée | 0 à 15 | obligatoire | Baudrate du composant :<br>• 2 : 500kbits/s<br>• 5 : 250kbits/s<br>Voir la spécification de la carte ESD pour les autres valeurs |
| Dialogue - Etats | | | | | |
| active | %integer | sortie | actif (1)<br>inactif (0) | optionnelle | Activée quand le composant émet des trames sur le CAN |
| compare | %integer | sortie | vraie (1)<br>fausse (0) | optionnelle | Lors d'une lecture des configurations, résultat de la comparaison entre les attributs d'entrée et de sortie de configuration notés (*) |
| done | %integer | sortie | actif (1) | optionnelle | Actif quand le composant |

| | | | | | |
|---|---|---|---|---|---|
| | | | inactif (0) | | effectue un traitement |
| msg_lost_counter | %integer | sortie | 0 à max %int | optionnelle | Compteur de trames perdues |
| msg_lost_alarm | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Timeout – 5 trames consécutives ont été perdues |
| frame_error_alarm | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | 5 trames consécutives comportent des erreurs (ID, RTR, longueur) |
| message_error | %integer | sortie | succès (0)<br>erreur (≠0) | optionnelle | Retour des fonctions du driver CAN. Ici, toutes les erreurs sont fatales et bloquent le composant (voir la spécification de la carte pour le détail des erreurs) |
| Contrôle - Entrées | | | | | |
| cons_freq | %real | entrée | -70 à 70 | obligatoire | Consignes en fréquence |
| clear_defaults | %integer | entrée | actif (1)<br>inactif (0) | obligatoire | Efface les défauts du variateur |
| isolation | %integer | entrée | non isolé (1)<br>isolé (0) | obligatoire | Isole le variateur par ouverture du relais du bus HT. Cette opération doit être réalisé avec vbus < 48V |
| Contrôle - Sorties | | | | | |
| temp_pt100 | %integer | sortie | 0 à 99 | optionnelle | Température de la sonde PT100 (non connectée = 99) |
| temp_igbt | %integer | sortie | 0 à 109 | optionnelle | Température des IGBT |
| ibus | %real | sortie | 0 à 30 | optionnelle | Courant du bus continu HT<br>Résolution 0,1A |
| vbus | %real | sortie | 0 à 500 | optionnelle | Tension du bus continu HT<br>Résolution 0,1V |
| i1_rms | %real | sortie | 0 à 35 | optionnelle | Courant phase U<br>Résolution 0,1A |
| i2_rms | %real | sortie | 0 à 35 | optionnelle | Courant phase V<br>Résolution 0,1A |
| i3_rms | %real | sortie | 0 à 35 | optionnelle | Courant phase W<br>Résolution 0,1A |
| rotation | %integer | sortie | antihoraire (1)<br>horaire (0) | optionnelle | Sens de rotation du moteur |
| sup_iphmax** | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Surintensité bus 280V<br>Provoque l'arrêt du variateur |
| inf_udcmin** | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Sous-tension bus 280V<br>Provoque l'arrêt du variateur |
| sup_udcmax** | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Surtension du bus 280V<br>Provoque l'arrêt du variateur |
| igbt_hitemp** | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Température haute IGBT<br>Provoque l'arrêt du variateur |
| iph_unstable | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Déséquilibre courant phase |
| freq_drawdown | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Rabattement de fréquence |
| pack_default** | %integer | sortie | vrai (1) | optionnelle | Défaut pack puissance |

| | | | | | |
|---|---|---|---|---|---|
| | | | faux (0) | | Provoque l'arrêt du variateur |
| soft_inhibition | %integer | sortie | active (1)<br>inactive (0) | optionnelle | Inhibition logicielle provoqué par un des 5 défauts bloquant noté (**) |
| vbus_sup_400 | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Tension bus HT > 400V |
| vbus_inf_95 | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Tension bus HT < 95V |
| ibus_sup_90 | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Courant bus HT > 90A |
| sup_uphnom | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Tension bus HT > uphnom |
| misc_defaults | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Défaut Alim BT<br>Inhibition matérielle<br>Défaut matériel ibus |
| i2c_error | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Erreur I2C |
| alarm_flag | %integer | sortie | vrai (1)<br>faux (0) | optionnelle | Une des alarmes (sup_iphmax à i2c_error) est présente |
| Configuration - Entrées | | | | | |
| fbase_in* | %integer | entrée | 30 à 70 | optionnelle | Fréquence de base de la loi de commande U/F (Hz) |
| fmax_in* | %integer | entrée | 30 à 70 | optionnelle | Fréquence Maximale de la loi U/F (Hz) |
| iphr_in* | %integer | entrée | 1 à 30 | optionnelle | Courant maximal autorisé par phase avant rabattement en fréquence |
| iphmax_in* | %integer | entrée | 1 à 35 | optionnelle | Courant efficace maximal autorisé par phase |
| acc_in* | %integer | entrée | 0 à 100 | optionnelle | Rampe positive de consigne fréquence (x 0,1s) |
| dec_in* | %integer | entrée | 0 à 100 | optionnelle | Rampe négative de consigne fréquence (x 0,1s) |
| udcmax_in* | %integer | entrée | 100 à 350 | optionnelle | Tension maximale du bus continu pour l'application |
| udcmin_in* | %integer | entrée | 100 à 350 | optionnelle | Tension minimale du bus continu pour l'application |
| uphnom_in* | %integer | entrée | 0 à 240 | optionnelle | Tension Ph/Ph paramétrable en fonction de UDCmin |
| npole_in* | %integer | entrée | 2, 4, 6, 8 | optionnelle | Nombre de pôles moteur |
| fdec_in* | %integer | entrée | 10 à 20 | optionnelle | Fréquence de découpage MLI (kHz) |
| sel_cna_in | %integer | entrée | 0 à 6 | optionnelle | Configuration de la sortie du CNA (0 par défaut) |
| c_clear_defaults_in | %integer | entrée | actif (1)<br>inactif (0) | optionnelle | Efface les défauts du variateur (mise à 0 auto.) |
| c_on_off_ihm_in | %integer | entrée | marche (1)<br>arrêt (0) | optionnelle | Marche / Arrêt de l'onduleur (1 par défaut) |
| start_write_in | %integer | entrée | vrai (1)<br>faux (0) | optionnelle | Sauvegarde les valeurs des registres du variateur en |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | mémoire I2C (mise à 0 auto.) |
| clear_rst_fpga_in | %integer | entrée | vrai (1)<br>faux (0) | optionnelle | Efface l'information Reset du FPGA (mise à 0 auto.) |
| sel_mode_in*** | %integer | entrée | numérique (1)<br>analogique (0) | optionnelle | Choix de la consigne en fréquence : numérique ou analogique (1 par défaut) |
| Configuration - Sorties | | | | | |
| fbase_out* | %integer | entrée | 30 à 70 | optionnelle | Fréquence de base de la loi de commande U/F (Hz) |
| fmax_out* | %integer | entrée | 30 à 70 | optionnelle | Fréquence Maximale de la loi U/F (Hz) |
| iphr_out* | %integer | entrée | 1 à 30 | optionnelle | Courant maximal autorisé par phase avant rabattement en fréquence |
| iphmax_out* | %integer | entrée | 1 à 35 | optionnelle | Courant efficace maximal autorisé par phase |
| acc_out* | %integer | entrée | 0 à 100 | optionnelle | Rampe positive de consigne fréquence (x 0,1s) |
| dec_out* | %integer | entrée | 0 à 100 | optionnelle | Rampe négative de consigne fréquence (x 0,1s) |
| udcmax_out* | %integer | entrée | 100 à 350 | optionnelle | Tension maximale du bus continu pour l'application |
| udcmin_out* | %integer | entrée | 100 à 350 | optionnelle | Tension minimale du bus continu pour l'application |
| uphnom_out* | %integer | entrée | 0 à 240 | optionnelle | Tension Ph/Ph paramétrable en fonction de UDCmin |
| npole_out* | %integer | entrée | 2, 4, 6, 8 | optionnelle | Nombre de pôles moteur |
| fdec_out* | %integer | entrée | 10 à 20 | optionnelle | Fréquence de découpage MLI (kHz) |
| sel_cna_out | %integer | entrée | 0 à 6 | optionnelle | Configuration de la sortie du CNA (0 par défaut) |
| c_clear_defaults_o | %integer | entrée | actif (1)<br>inactif (0) | optionnelle | Efface les défauts du variateur (doit être lu à 0) |
| c_on_off_ihm_out | %integer | entrée | marche (1)<br>arrêt (0) | optionnelle | Marche / Arrêt de l'onduleur (1 par défaut) |
| start_write_out | %integer | entrée | vrai (1)<br>faux (0) | optionnelle | Sauvegarde les valeurs des registres du variateur en mémoire I2C (doit être lu à 0) |
| clear_rst_fpga_out | %integer | entrée | vrai (1)<br>faux (0) | optionnelle | Efface l'information Reset du FPGA (doit être lu à 0) |
| sel_mode_out*** | %integer | entrée | numérique (1)<br>analogique (0) | optionnelle | Choix de la consigne en fréquence : numérique ou analogique (1 par défaut) |

(***) La consigne en fréquence est forcée en numérique par la carte interface

(**) Alarmes bloquantes provoquant l'arrêt du variateur

(*) Configurations du variateur pouvant être sauvegardées en mémoire I2C.

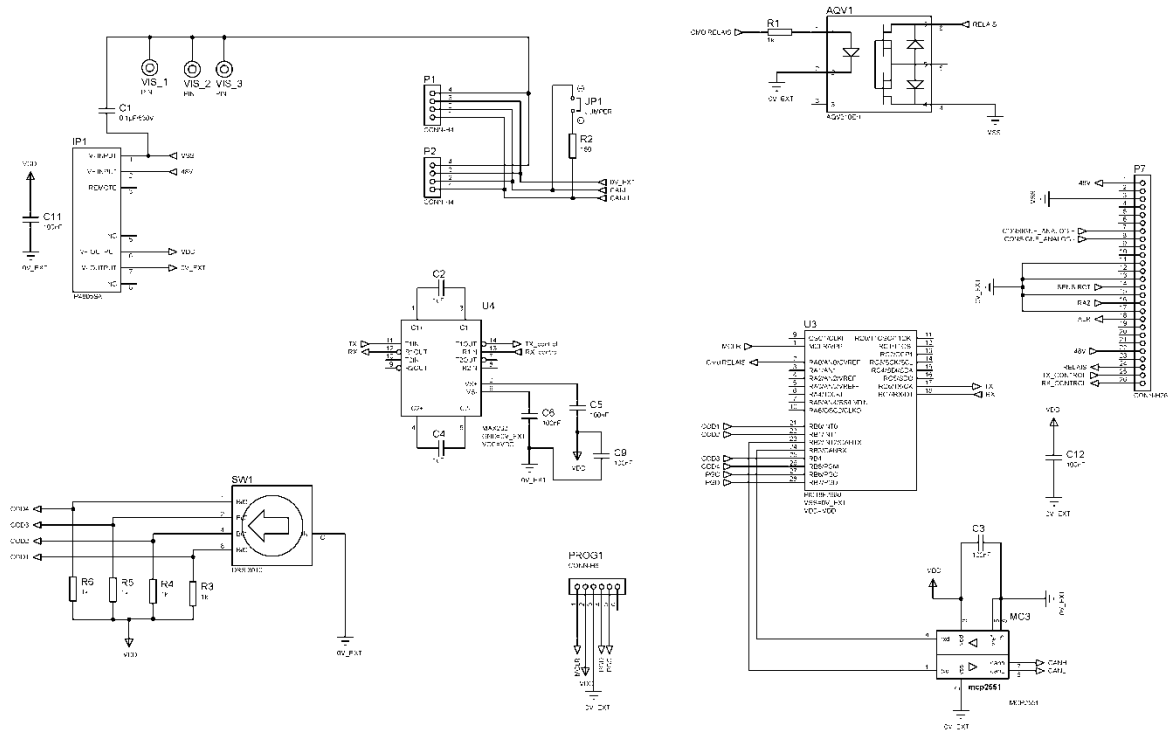
## 11.2. Liste des alarmes de la carte ESD CAN-CPCI/331-2

Cette liste d'alarmes est le retour des fonctions C++ du driver CAN. Le numéro d'erreur est signalé sur l'attribut "error_message" du composant de pilotage.

| Code d'erreur | numéro | Signification | Résultat |
|---|---|---|---|
| NTCAN_SUCCESS | 0 | Opération réussie | - |
| NTCAN_RX_TIMEOUT | 0x00001001 | Pas possible car asynchrone | Erreur fatale |
| NTCAN_TX_TIMEOUT | 0x00001002 | Pas possible car asynchrone | Erreur fatale |
| NTCAN_TX_ERROR | 0x00001004 | Pas possible car asynchrone | Erreur fatale |
| NTCAN_CONTR_OFF_BUS | 0x00001005 | Pas possible car asynchrone | Erreur fatale |
| NTCAN_CONTR_BUSY | 0x00001006 | Pas possible car asynchrone | Erreur fatale |
| NTCAN_CONTR_WARN | 0x00001007 | Pas possible car asynchrone | Erreur fatale |
| NTCAN_NO_ID_ENABLED | 0x00001009 | Le canID n'a pas été initialisé | Erreur fatale |
| NTCAN_ID_ALREADY_ENABLED | 0x0000100A | Problème d'init des canID | Erreur fatale |
| NTCAN_ID_NOT_ENABLED | 0x0000100B | Pas possible car CanIdDelete() n'est pas utilisé | Erreur fatale |
| NTCAN_INVALID_FIRMWARE | 0x0000100D | Problème carte ESD | Erreur fatale |
| NTCAN_MESSAGE_LOST | 0x0000100E | Pas dans la doc !! | Erreur fatale |
| NTCAN_INVALID_HARDWARE | 0x0000100F | Problème carte ESD | Erreur fatale |
| NTCAN_PENDING_WRITE | 0x00001010 | Ne devrait jamais arriver car 1 handle par thread est utilisé | Erreur fatale |
| NTCAN_PENDING_READ | 0x00001011 | Ne devrait jamais arriver car 1 handle par thread est utilisé | Erreur fatale |
| NTCAN_INVALID_PARAMETER | EINVAL (22) | Paramètre invalide dans le call, logiquement pas possible | Erreur fatale |
| NTCAN_INVALID_HANDLE | EINVAL (22) | handle invalide (problème des timers par ex.) | Erreur fatale |
| NTCAN_NET_NOT_FOUND | ENODEV | Interface CAN pas trouvé lors du canOpen() | Erreur fatale |
| NTCAN_INSUFFICIENT_RESOURCES | ENOMEM | Trop de handle | Erreur fatale |

## 11.3. Préconisations pour la carte interface

### 11.3.1. Schéma de la carte




### 11.3.2. Analyse et modifications de la carte interface

Avec le STR, il apparaît quelques erreurs de transmission : trames perdues, erreurs de données sur le Buscan qui m'ont poussé à investiguer l'électronique et la programmation de la carte interface.

**Software :**

Développement d'un nouveau programme de gestion séquentielle avec ajout de la prise en compte des erreurs de liaison série P+ (ACK, NACK, CHKSUM, TIMEOUT) lors de la scrutation des registres du variateur. Une reprise d'erreur est donc possible et le type d'erreur est signalé sur le Buscan pour Débug.

Sans puissance, il apparaît de nombreux TIMEOUT. Avec oscilloscope, il est montré que la carte variateur ne répond pas à toutes les interrogations. Dans un premier temps, les temporisations de dialogue sont allongées mais ne permettent pas de supprimer cette erreur.

Le problème est résolu en ajoutant un quart 8MHz car le RC interne du microcontrôleur dérive. Le microcontrôleur chauffe légèrement et le RC dérive, entraînant des erreurs de débit sur la liaison série et sur le Buscan à long terme. Il est d'ailleurs préconisé par Microchip d'utiliser un quartz avec des débits supérieurs à 125kbits/s sur le Buscan.

**Hardware :**

Les condensateurs autours du MAX232 ont des valeurs trop faibles, les 100nF doivent être relevées au moins à 1uF. Ces condensateurs sont polarisés : la sérigraphie présente une erreur sur C2, le condensateur était monté à l'envers.

Les condensateurs de découplage 100nF tantale étaient tous montés à l'envers : le câbleur a du se servir des empreintes des condensateurs polarisés pour monter ceux-ci (broche + dans le trou carré). Ils ont été remplacés par des 100nF non polarisés.

Le régulateur manque de filtrage. J'ai ajouté un condensateur réservoir de 10uF en sortie et il serait nécessaire d'appliquer un gros filtrage en entrée sur le bus 48V.

**Essais en bassin :**

La communication est très bonne sans la puissance.

Avec puissance, quelques pertes de trames BusCAN apparaissent mais qui permettent tout de même un fonctionnement correct de cette partie de dialogue. Sur la liaison série, quand le variateur fonctionne à vitesse constante et ne reçoit pas d'ordre changement de vitesse, on compte une à deux trames perdues par seconde, soit 1 à 2 trames pour 147 trames (7ms env. par cycle). Lors des changements de vitesse, à basse vitesse et lors des changements du sens de rotation, jusqu'à 16 trames consécutives peuvent être perdues sur la liaison série, soit 110ms pendant lesquelles aucun dialogue n'aboutit vers le variateur.

**Ajout plan de masse :**

Un plan de masse (au châssis), découpé dans un PCB, est ajouté sous la carte interface. Il n'y a plus de perte de trames sur la liaison série. Il en reste quelques-unes sur le BusCAN mais moins fréquentes que précédemment.

## 11.4. Notes des compilations STR

sur la machine de développement - C:\VICTOR en partage
ce répertoire est monté sur le STR - /victor
mon répertoire de travail - C:\VICTOR\LTDev

**********  récupérer depuis le serveur de version SVN **********
------- récupérer une version d'ACE
créer le rep si c'est une nouvelle version : r1.4b1 par ex
depuis le répertoire C:\VICTOR\LTDev\ace\r1.4b1 click droit -> SVN extraire
URL du référentiel: file:///W:/PROJETS/VICTOR/SVNAceVictor/ace/branches/r1.4b1
extraction du répertoire: C:\VICTOR\LTDev\ace\r1.4b1
file:///W:/PROJETS/VICTOR/SVNAceVictor/cfg/branches/b29

------- de même pour CFG
créer le rep si c'est une nouvelle version : b29 par ex
depuis le répertoire C:\VICTOR\LTDev\cfg\b29 click droit -> SVN extraire
URL du référentiel:file:///W:/PROJETS/VICTOR/SVNAceVictor/cfg/branches/b29
extraction du répertoire:C:\VICTOR\LTDev\cfg

------- pour effectuer un commit (SVN Livrer)
Depuis : C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\export\lib
click droit sur : victor.lib -> commit
faire pareil avec : C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\export\bin
click droit sur : ace_victor -> commit

**********  infos sur CFG : les feuilles Protel *.sch **********
les feuilles Protel *.sch sont compilées sous Protel icône SCH -> ACE
-> .evt (signaux)
-> .cmp (lien signaux-composants)

**********  recompiler ACE **********
------- les lib
C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\victor.lib\build
il faut compiler à partir du STR : /victor/LTDev/ace/r1.4b1Dev/ace/victor.lib/build
make puis make export
le victor.lib est exporté dans : C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\export\lib

------- ACE
C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\ace.exe\build
à partir du STR : /victor/LTDev/ace/r1.4b1Dev/ace/ace.exe/build
make puis make export
le victor.lib est exporté dans : C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\export\bin

--------- copie des fichiers
après la compilation d'ACE, il faut copier les fichiers sur le STR
copie des répertoires bin et lib de C:\VICTOR\LTDev\ace\r1.4b1Dev\ace\export
STR : /victor/LTDev/ace/r1.4b1Dev/ace/export/

/usr/local/ace_victor/r1.4b1/

le repértoire r1.4b1 de la nouvelle version doit être créé s'il n'existe pas

depuis le rep : /usr/local/ace_victor/r1.4b1/
exécuter : cp -R /victor/LTDev/ace/r1.4b1Dev/ace/export/ .

------- mettre les bons droits
chmod 4555 /usr/local/ace_victor/r1.4b1/bin/ace_victor

------- pour travailler sur les fichiers config de la machine de développement
sur le STR : /home/victor
lien symbolique : ln -sf /victor/LTDev/cfg/b29dev/ cfg
sinon la version officielle : ln -sf /home/victor/rel/r1.3b27 cfg

------- mettre les bons droits
dans /victor/LTDev/cfg : chmod 4555 gw etc…

------- pour pointer sur la dernière version d'ACE
sur le STR : /usr/local/bin
lien symbolique : ln -sf /usr/local/ace_victor/r1.4b1/bin/ace_victor ace_victor

**********  **sur le temps réel**  **********
ne pas oublier de taper C331 !!!! pour init le driver CAN
car il n'est pas activé par défaut sur toutes les machines

**********  **GUI**  **********
dans Phindows, onglet Dev. Tools (en bas), lancer PhAB

on peut donc ajouter des Widgets (Label, Button, etc...).
Le nom du Widget doit correspondre avec le nom déclaré dans le fichier .ini correspondant.

Pour une copie de Wigdet: CTL + clic gauche et déplacer le widget à l'endroit désiré
et ne pas oublier Code:Realized

Il est impératif pour un nouveau Widget d'ajouter dans Callbacks (bas gauche)
Realized -> Code:Realized
Dans la fenêtre qui s'ouvre, ajouter le nom de la fonction (link to callback/module info)
"Realized@isewa.c"

Pour compiler : Application -> Build/Run (F5)
Puis Generate et Make
sinon aller dans le rep. /home/victor/gui/1.2b18dev/victor_gui/src/default et faire make

puis il faut faire la copie de victor_gui et victor_gui.map qui ont été générés
dans /usr/local/victor_gui/1.2b18dev
cp /home/victor/gui/1.2b18dev/victor_gui/src/default/victor* .

puis les droits : chmod 4555 victor_gui victor_gui.map
et faire le lien sur la nouvelle version :
ln -sf /usr/local/victor_gui/1.2b18dev/victor_gui victor_gui

## 11.5. Notes des modifications des feuilles Protel

**// ** Vérification de Tension Basse (low_umin)**
umin -> low_umin
dans: alarm_message_manager_1.sch
460 hits

**// ** Vérification de Tension Haute (high_umax)**
"umax" -> high_umax
dans: alarm_message_manager_1.sch
460 hits

**// ** Vérification de Température (temperature)**
pas possible car d'autres alarmes de température existent : confiance car aucun changement

**// ** Vérification de Rabattement de fréquence (freq_pb)**
460 hits

**// ** Vérification de Tension Haute (high_umax)**
"pow_pack" -> power_pack
Dans : alarm_message_manager_4.sch
460 hits

**// ** Internal Link devient Sur-intensité**
internal_link -> sup_iphasemax
int_link -> sup_iphasemax
460 hits OK

--- correction erreur dans:
Button ButtonHydMotorSupIphmaxInhib
{
  //control_event =
hyd_motor_ala_**temperature**_inhib;
        **//erreur**
  control_event =
hyd_motor_ala_sup_iphasemax_inhib;
  off_fill_color  = red;
}

alarme_defaut_liaison -> alarme_surintensite
dans: gui_thruster_interface.ini

InternalLink -> SupIphmax
dans: gui_alarms.ini thrusters.ini

**// ** 48V sous tension devient Courant instable**
elec_pb -> iph_uns
460 hits OK

48v_sous_tension -> courant_instable
dans: gui_thruster_interface.ini

ElecPb -> IphUnst
dans: gui_alarms.ini thrusters.ini

**// ** Iso State devient Defaut Divers**
l'alarme iso_state n'est pas câblée....

ajout : iso_state
dans: alarms_thrusters_1.sch

cette alarme n'est plus dispo, 7 occurrences supprimées dans: gui_thruster_interface.ini
BooleanLabel PtLabel_alarme_etat_relais_xxx
et ajout de 7 occurrences
BooleanLabel PtLabel_alarme_defaut_divers_xxx

ajout : iso_state
dans: alarms_disable_manager.sch

création de message_manager_6.sch pour déplacer :
alarm_msg_active_9
alarm_msg_active_10
alarm_msg_active_11
qui étaient dans message_manager_4.sch

ajout :iso_state
dans: message_manager_4.sch
sur le groupe: alarm_msg_active_8

ajout : iso_state
dans: gui_alarms.ini
14 occurrences
BooleanLabel ButtonThrusterxxxMiscDefault
Button ButtonThrusterxxxMiscDefaultInhib

ajout : iso_state
dans: logging_asynchro_inhib_3.sch
35 occurrences
ajout : iso_state
dans: thrusters.ini
21 occurrences
BooleanLabel ButtonThrusterxxxMiscDefault
Button ButtonThrusterxxxMiscDefaultInhib

et modification des "alarm_text"
432 hits + 28 (compenvent + .old) = 460 OK

/**************************************
*  MENAGE
**************************************/
Démontage de l'ancien composant moteur dans:
thruster_interface.sch

suppresion des persists
suppresion du persist Setup dans: hci.sch

démontage des sorties de config (fmli, acc, dec , ili, temp, ala_22) et réutilisation dans:
thrusters_status.sch

| | | |
|---|---|---|
| fmli_out | -> | NU |
| acc_out | -> | cfg_out_acc |
| dec_out | -> | cfg_out_dec |
| ili_out | -> | cfg_out_iphmax |
| temp_out | -> | out_temp_igbt |
| ala_22 | -> | NU |

démontage des états (ic, uc, ala_44)
et réutilisation dans: thrusters_status.sch

| | | |
|---|---|---|
| amps | -> | out_ibus |
| volts | -> | out_vbus |
| ala_44 | -> | NU |

suppression de: status_ala_44
dans : logging.sch

démontage des consignes
démontage de la liaison série
suppression des paramètres série dans :
setports_24

```
# Settings for Thruster control
# File : thruster_interface.cmp
stty +raw -isflow -osflow -ihflow -ohflow -lkhflow
baud=19200 < /dev/ex9
echo 'Thruster settings...'
stty < /dev/ex9
echo
```

démontage des send_frame
démontage du composant victor_thrusters

copie du reste de: thruster_interface.sch
dans: thruster_can_interface.sch
et suppression de: thruster_interface.sch

/**************************************
*  SCRIPTS
**************************************/
ajout de clear_default automatique
dans : thrusters_simu_varpackxxx.sch

les 5 défauts les plus importants (surintesité, sous tension, sur tension, temp, défaut pack)
sont câblés dans un "OU" pour faire passer la consigne à 0, ainsi lors de l'acquitement, le clear défaut sera passé avec des consignes à 0 et le fonctionnement devrait pouvoir reprendre avec le défaut supprimé

le script d'acquittement des alarmes est supprimé:
thrusters_alarm_ack.bsn
et remplacé par du schéma (activation de 1s du clear défaut)

modification de: procedures_M_A_1.sch
procedures_M_A_2.sch
alarm_ack.sch

pour changer renommer thruster_alarm_ack et inclure le "NOT"
dans: procedures_M_A_2.sch

script d'init est modifié:
il ne faut pas faire mettre thrusters_stop_enable = 0 en fin de script et laisser ceci à la gestion des passages par 0 des consignes -> il faudra donc (ré)initialiser le manche pilote à l'init

la demande de config est ajoutée dans le script

/**************************************
*  victor lib
**************************************/
la demande de config est supprimée de VictorCanThruster.cpp, c'est à dire qu'il n'y a plus de demande config on rentre dans le mode consignes

_message_error ne repassait pas à 0 après enable:
une mise à jour de cette broche a été ajoutée à la fin de la méthode Subscribe".

/**************************************
*  GUI
**************************************/
modification des widgets de thruster_interface dans le GUI

48v_sous_tension -> courant_instable
dans: gui_thruster_interface.ini

alarme_defaut_liaison -> alarme_surintensite
dans: gui_thruster_interface.ini

iso_state -> alarme-defaut_divers
dans: gui_thruster_interface.ini

Suppression de tous les Warnings en modifiant/ajoutant les widgets en concordance avec les noms dans les fichiers:
gui_alarms.ini
gui_thruster_interface.ini

/**************************************
*  ajout de l'alarme BUSCAN
**************************************/
dans: alarm_ack.Sch
ajout de: thrusters_buscan_global_alarm

dans: fault_manager.sch
ajout de thrusters_buscan_global_alarm sur fault 26 et modification du fichier "victor_frt.csv" pour les missions
ATTENTION: ne pas éditer ce fichier avec excel !!!

création de : alarms_digital_3.sch
pour regrouper l'alarme Bus CAN, au lieu d'avoir des petits bouts dans:
alarm_digital
logging_asynchro_inhib
alarm_disable_manager

création de : message_manager_6.sch
message_manager_7.sch
pour alléger les message_manager_1_2_3_4_5.sch

l'alarme Bus CAN est ajoutée dans:
message_manager_5.sch

/**************************************
*  Nettoyage: la suite
**************************************/
suppresion de "thrusters_frame_44_alals" dans :
logging.sch
thruster_can_inteface.sch

suppression de "thrusters_frame_22_alals" dans :
thruster_can_inteface.sch

suppression de "thrusters_frame_22_count" dans :
thruster_can_inteface.sch

suppression de "thrusters_frame_44_count" dans :
thruster_can_inteface.sch

supprimé de : gui_thruster_interface_ini.sch
Label label_frame_22_alarm {…}
Label label_frame_22_count {…}
Label label_frame_44_count {…}

---
suppression du bloc de composants "thruster_write" dans : thruster_can_inteface.sch
qui était non utilisé

supprimé de: gui_thruster_interface_ini.sch
#MomentaryButton button_thruster_write_cfg {…}

---
suppression du bloc de composants "thruster interface timeout" dans :
thruster_can_inteface.sch
qui n'est plus utilisé avec thrsuter CAN

supprimé de: persist.txt
thrusters_frames_rcvd_timeout_ms_ui 7000

suppression de "thrusters_interface_fault" dans :
alarms_digital.sch

qui amène à supprimer aussi:
"thrusters_interface_alarm"
"thrusters_interface_alarm_ack" dans :
alarm_ack.sch

supprimé de: gui_alarms.ini
AlarmButton ButtonThrusterInterfaceFault {…}
Button ButtonThrusterInterfaceFaultInhib {…}

suppresion dans :
alarm_disable_manager.sch
logging_asynchro_inhib_1.sch
des blocs correspondant à
“thruster_interface_alarm"

dans : fault_manager.sch
suppression de "thrusters_interface_alarm" sur fault_12 et déplacement de "thrusters_buscan_global_alarm" sur fault 12
modification du fichier "victor_frt.csv" pour les missions

ATTENTION: ne pas éditer ce fichier avec excel !!!

Dans : procedures_M-A_1.sch
shuntage du bloc "thruster_free_cmd_trigger"

suppresion de "Perte liaison moteur" dans le GUI : Alarm_thruster
car "thrusters_interface_alarm" n'existe plus

/**************************************
*  ajout de l'alarme Msg Lost
**************************************/
dans: alarm_ack.Sch
ajout de: thrusters_msg_lost_global_alarm

dans : fault_manager.sch
ajout de thrusters_buscan_global_alarm sur fault 26 et modification du fichier "victor_frt.csv" pour les missions
ATTENTION: ne pas éditer ce fichier avec excel !!!

création de : alarms_digital_4.sch
pour regrouper l'alarme Msg Lost, au lieu d'avoir des petits bouts dans:
alarm_digital
logging_asynchro_inhib
alarm_disable_manager

l'alarme Msg Lost est ajoutée dans:
message_manager_5.sch

/**************************************
*  Test de la carte variateur et interface
**************************************/
OK pour les init du dialogue

pb affichage sur les short -> inversion des poids/faible dans le PIC

pb affichage sur les alarmes -> correction des OU et des masques

pb sur la température -> réecriture du calcul en détaillant

passage en double 32 bits (sqrt)
ça ne fonctionne pas avec le débugueur (conflit mem.)

l'alarme soft_inhibition est commune aux 5 alarmes bloquantes -> modification de thruster_can_interface.sch et thruster_satus.sch pour essai sur thruster_1

/**************************************
*  Ajout compteur trames perdues
**************************************/
modifications de victor_can_thruster.cpp pour ajout broche de comptage des messages perdus.
Au bout de 5 messages perdus consécutivement, l'alarme msg_lost_error passe à 1 et le composant est bloqué. Il faut ré-initialiser par un disable/enable, sinon stopper, lire la config et go.

ajout dans: gui_thruster_interface_ini.sch
Label label_thruster_1_msg_lost {…}
etc...

/**************************************
*  ajout de l'alarme Msg Lost
**************************************/
Dans : alarm_ack.Sch
ajout de: thrusters_msg_lost_global_alarm

dans : alarms_digital_2.sch
ajout de: thrusters_msg_lost_counter_fault

l'alarme Msg Lost est ajoutée dans:
message_manager_5.sch

mise en commun de :
un "OU" est câblé dans alarm_ack.sch pour regrouper les alarmes de liaison CAN: thruster_interface_global_alarm_ack (dans alarm_ack)
L'alarme message lost ne détaille pas quel variateur est en perte de trame, il faut aller regarder les compteurs.

/**************************************
*  Arrangement des alarmes
**************************************/
thrust_X_ctl_out_alarm_flag est un "OU" de toutes les alarmes -> il n'a pas besoin d'être câblé dans thruster_status.sch

thrust_X_ctl_out_soft_inhibition
SOFT_INHIBITION est un "OU" des 5 alarmes bloquantes: on peut donc remplacer les 5 alarmes dans thruster_can_interface par cette alarme et faire un ou des 5 alarmes dans les simulateurs (dans le .cpp)

/**************************************
*  modification de victor_can_thrusters.cpp
**************************************/
les modifs ne sont pas prises en compte à la compil

dans le répertoire:
C:\ACE_VICTOR\LTDev\ace\r1.4b1Dev\ace\ace.exe\build

il existait une version de victor.lib dont la date était en 2083. La RTC du STR doit avoir ses piles HS: attention aux problèmes de dates.

//------- reprise des modifs
et il faut que MsgLost et Timeout soit remis à 0 si on refait une init. PQ ça marche pas ?
-> il fallait une tempo entre "enable" et "go" (300ms) dans le script init

modifier le code quand il y a un timeout pendant les config... c'est pas clair
_wrState = WR_ERROR; au lieu de _mState = WR_ERROR;
-> OK corrigé, c'est bien mieux !!

il ne faut pas qu'il y ait de timeout durant les read config... A TESTER

ajouter un case pour traiter l'erreur de timeout pendant les config
```
case M_TIMEOUT_ERROR:
{
        DebugMsg( STATE_MACHINE, fName,
        "Timeout bla bla");
        _mState = M_ERROR_NO_MESSAGE;
}
```

/**************************************
*  modification de message lost counter
**************************************/
cette alarme globale n'était pas désactivable -> ajout d'un copy dans "alarms_digital_2.sch"

elle n'était pas visible dans les alarmes variateurs
-> modification du widget en "PtButton" au lieu de PtLabel
modifier l'alarme message lost counter pour la paramétrer toutes les 10, 100, 1000 trames:
-> ajout boutons de paramètrage dans le GUI
-> ajout d'un input_select dans "thrusters_status.sch"

/**************************************
*  ajout de Write Config
**************************************/
un scrit a été ajouté pour écrite les configuration: "thrusters_write.bsn"

dans le composant victor_can_thruster
modifications:
la broche "compare" ne teste pas le registre CONF
la broche "active" n'indique plus quand le buscan est actif, mais quand le composant envoie des trames

pour écrire les configs, le composant peut être en RUN ou STOP et les consignes doivent être à 0. Après écriture, le composant sera en STOP, une comparaison des configs aura été effectuée.

/**************************************
*  GUI thruster_interface
**************************************/
on a décidé de ne pas afficher les configs dans l'IHM mais que les états car ils sont inutiles pour l'utilisateur.

On affichera que des états: Ubus, Ibus, Temp IGBT, I1rms, I2rms et I3rms.

Ajout de :"thruster_status_2.sch"
pour afficher un message:

| | |
|---|---|
| RUN | envoie des trames |
| STOP | pas d'envoie en cours (stop, timeout, buscan error, etc...) |
| Config OK | config correspond aux persists |
| Config Non OK | ne correspond pas aux persists |
| Inactif | disable |

/**************************************
*  barometers
**************************************/
En construction.....

dans hci.sch
ajout d'un composant de déclaration de persist pour persist_barometers.dat
ajout dans : persist_db_restore
/usr/local/bin/perctl -f /home/victor/persist_barometers.dat -a /home/victor/cfg/persist_barometers.txt;

ajout de persist_barometers.txt

ajout de barometers.sch dans
manager_rov_sensors.sch

/**************************************
*  Nettoyage
**************************************/

les "net" suivants :
hyd_motor_speed_dirn
hyd_motor_speed_hz
étaient définies dans thruster_can_interface.sch
et ne sont pas utilisées dans cette page. Ils sont
définis dans procedures_M_A_2.sch, d'où ils
proviennent et supprimés de :
thruster_can_interface.sch

---
quand on recopie vers le STR:
"manager_rov_sensors.*", il sort des erreurs
car il faut désactiver:
//%include file="phins_halliburton.cmp"
//%include file="phins_gga_input.cmp"
//%include file="phins_halliburton.evt"
//%include file="phins_gga_input.evt"
dans manager_rov_sensors.evt
manager_rov_sensors.cmp
avant la copie

finalement, on a décidé d'enlever
phins_halliburton.sch du mananger
et de laisser activer phins_gga_input.sch car il ne
pose plus de problèmes au niveau du réseau
---
les persists des thrusters et de simu_varpack sont
regroupées en 2 fichiers:
persist_can_thrusters.txt
persist_simu_varpacks.txt

/**************************************
*  A FAIRE si nécessaire
**************************************/
lookup table dans thruster_sp_management.sch
associées à :
thrusters_setpoints.txt
thrusters_return.txt
explications: permet de faire les quotas avec les
tables. Comme les tables ne sont pas linéaires,
la vitesse (Hz) est relinéarisée avant envoie vers les
variateurs.

dans victor.lib:
supprimer l'ancien composant victor_thrusters.cpp
et les include :
#include "victor_frame_t66.h
#include "victor_frame_t88.h
#include "victor_frame_t11.h
#include "victor_frame_t22.h
#include "victor_frame_t44.h