IgH EtherCAT® Master 1.5.2 Documentation

Dipl.-Ing. (FH) Florian Pose, fp@igh.de
Ingenieurgemeinschaft *IgH*

Essen, 6 octobre 2025
Révision 1.6.8

Traduit en français par Sébastien BLANCHET

Table des matières

Liste des tableaux

Table des figures

Conventions

Ce document utilise les conventions typographiques suivantes :

- Le *texte en italique* est utilisé pour introduire des nouveaux termes et pour les noms de fichiers.
- Le `texte à chasse fixe` est utilisé pour les exemples de code et les sorties des lignes de commandes.
- Le `texte en gras à chasse fixe` est utilisé pour les entrées utilisateurs dans les lignes de commandes.

Les valeurs des données et des adresses sont habituelles spécifiées en valeurs hexadécimales. Elles sont indiquées dans le style du langage de programmation *C* avec le préfixe `0x` (par exemple : `0x88A4`). Sauf mention contraire, les valeurs des adresses sont spécifiées en adresse d'octets.

Les noms des fonctions sont toujours écrits avec des parenthèses, mais sans paramètre. Ainsi, si une fonction `ecrt_request_master()` a des parenthèses vides, ceci n'indique pas qu'elle ne prend pas de paramètres.

Les commandes shell à taper, sont indiquées par un prompt dollar :

```
$
```

Par ailleurs, si une commande shell doit être tapée en tant que le super utilisateur, le prompt est un dièse :

```
#
```

1 Le maître EtherCAT IgH

Ce chapitre couvre les informations générales à propos du maître EtherCAT.

1.1 Résumé des fonctionnalités

La liste ci-dessous donne un bref résumé des fonctionnalités du maître.

- Conçu en tant que module noyau pour Linux 2.6 / 3.x.
- Implémenté suivant la norme IEC 61158-12 [2] [3].
- Fourni avec des pilotes natifs EtherCAT pour plusieurs périphériques Ethernet courants, mais aussi avec un pilote générique pour toutes les puces Ethernet supportées par le noyau Linux.
  - Les pilotes natifs gèrent le matériel sans interruption.
  - Des pilotes natifs pour d'autres périphériques Ethernet peuvent être facilement implémentés en utilisant l'interface commune des périphériques (voir section 4.6) fournie par le module maître.
  - Pour les autres matériels, le pilote générique peut être utilisé. Il utilise les couches basses de la pile réseau de Linux.
- Le module maître supporte l'exécution en parallèle de plusieurs maîtres EtherCAT.
- Le code du maître supporte n'importe quelle extension temps réel de Linux au travers de son architecture indépendante.
  - RTAI [11] (y compris LXRT via RTDM), ADEOS, RT-Preempt [12], Xenomai (y compris RTDM), etc.
  - Il fonctionne aussi sans extension temps réel.
- Une "API" commune pour les applications qui veulent utiliser les fonctionnalités EtherCAT (voir chapitre 3).
- Des *domaines* sont ajoutés, pour permettre de grouper les transferts de données des processus avec différents groupes d'esclaves et de périodes des tâches.
  - Gestion de domaines multiples avec différentes périodes de tâches.
  - Calcul automatique de la cartographie des données des processus, FMMU et configuration automatique des gestionnaires de synchronisation au sein de chaque domaine.
- Communication au travers de plusieurs automates.
  - Analyse automatique du bus après les changements de topologie.
  - Surveillance du bus pendant les opérations.
  - Reconfiguration automatique des esclaves (par exemple après une panne d'alimentation) pendant les opérations.

- Support des horloges distribuées (Distributed Clocks)(voir section 3.5).
  - Configuration des paramètres d'horloges distribuées de l'esclave via l'interface de l'application.
  - Synchronisation (compensation du décalage et de la dérive) des horloges distribuées des esclaves avec l'horloge de référence.
  - Synchronisation optionnelle de l'horloge de référence avec l'horloge maître ou dans l'autre sens.
- CANopen over EtherCAT (CoE)
  - Téléversement, téléchargement et service d'information SDO.
  - Configuration des esclaves via SDOs.
  - Accès SDO depuis l'espace utilisateur et depuis l'application.
- Ethernet over EtherCAT (EoE)
  - Utilisation transparente des esclaves EoE via des interfaces réseaux virtuelles.
  - Support natif des architectures réseaux EoE commutées ou routées.
- Vendor-specific over EtherCAT (VoE)
  - Communication avec les boîtes aux lettres spécifiques des vendeurs via l'API.
- File Access over EtherCAT (FoE)
  - Chargement et enregistrement des fichiers via l'outil en ligne de commande.
  - La mise à jour du firmware de l'esclave peut être faite facilement.
- Servo Profile over EtherCAT (SoE)
  - Implémentation conforme à IEC 61800-7 [16].
  - Enregistrement des configurations IDN, qui sont écrites dans l'esclave pendant le démarrage.
  - Accès aux IDNs via l'outil en ligne de commande.
  - Accès aux IDNs pendant l'exécution via la bibliothèque en espace utilisateur.
- Outil en ligne de commande "ethercat" dans l'espace utilisateur (voir section 7.1)
  - Information détaillée à propos du maître, des esclaves, domaines et configuration du bus.
  - Paramétrage du niveau de déverminage du maître.
  - Lecture/Ecriture des adresses d'alias.
  - Listage des configurations des esclaves.
  - Affichage des données des processus.
  - Téléchargement/Téléversement SDO ; listage des dictionnaires SDO.
  - Chargement et enregistrement de fichiers via FoE.
  - Accès IDN SoE.
  - Accès aux registres des esclaves.
  - Accès à la SII (EEPROM) de l'esclave.
  - Contrôle des états de la couche application.
  - Génération de la description des esclaves au format XML et code C pour les esclaves existants.
- Intégration système transparente au travers de la conformité LSB.
  - Configuration du maître et des périphériques réseaux via des fichiers sysconfig.
  - Script d'initialisation pour le contrôle du maître.

— Fichier de service pour systemd.
— Interface réseau virtuelle en lecture seule pour la surveillance et le déverminage.

1.2 License

Le code source du maître est publiée selon les termes et conditions de la GNU General Public License (GPL [4]), version 2. Les développeurs, qui veulent utiliser EtherCAT pour les systèmes Linux, sont invités à utiliser le code source du maître ou même à participer à son développement.

Pour autoriser la liaison statique d'une application en espace utilisateur avec l'API du maître (voir chapitre 3), la bibliothèque pour l'espace utilisateur (voir section 7.2) est publiée selon les termes et conditions de la GNU Lesser General Public License (LGPL [5]), version 2.1.

2 Architecture

Le maître EtherCAT est intégré au noyau Linux. C'était une décision originelle de conception, qui a été prise pour plusieurs raisons :

— Le code du noyau a des caractéristiques de temps réel significativement meilleures, i. e. une latence plus faible que le code de l'espace utilisateur. Il était prévisible, qu'un maître pour un bus de terrain, ait beaucoup de travail cyclique à faire. Le travail cyclique est habituellement déclenché par des interruptions de timer dans le noyau. Le délai d'exécution d'une fonction qui traite une interruption de timer est moindre si elle réside dans l'espace noyau, parce qu'il n'y a pas besoin de passer du temps à commuter le contexte vers le processus en espace utilisateur.

— Il était prévisible, que le code du maître doive communiquer directement avec le matériel Ethernet. Ceci doit être fait dans le noyau de toute façon (au travers des pilotes des périphériques réseau), ce qui constitue une raison supplémentaire pour que le code du maître soit dans l'espace du noyau.

La figure 2.1 fournit une vue d'ensemble de l'architecture du maître.

Les composants de l'environnement du maître sont décrits ci-dessous :

Master Module Module noyau contenant une ou plusieurs instances du maître EtherCAT (voir section 2.1), le "Device Interface" (interface du périphérique, voir section 4.6) et l'"Application Interface" (interface de programmation applicative, voir chapitre 3).

Device Modules Modules de pilotes de périphérique Ethernet supportant EtherCAT qui offrent leurs périphériques au maître EtherCAT via l'interface du périphérique (voir section 4.6). Ces pilotes réseaux modifiés peuvent gérer en parallèle les interfaces réseaux utilisées pour les opérations EtherCAT et les interfaces réseaux Ethernet "normales". Un maître peut accepter un périphérique particulier pour envoyer et recevoir des trames EtherCAT. Les périphériques Ethernet déclinés par le module maître sont connectés comme d'habitude à la pile réseau du noyau.

Application Un programme qui utilise le maître EtherCAT (habituellement pour un échange cyclique de données de processus avec les esclaves EtherCAT). Ces programmes n'appartiennent pas au code du maître EtherCAT [1], mais ils doivent être générés ou écrits par l'utilisateur. Une application peut demander un maître via l'API (voir chapitre 3). Si la demande réussie, elle a alors le

1. Toutefois, il y a des exemples fournis dans le dossier *examples/*.

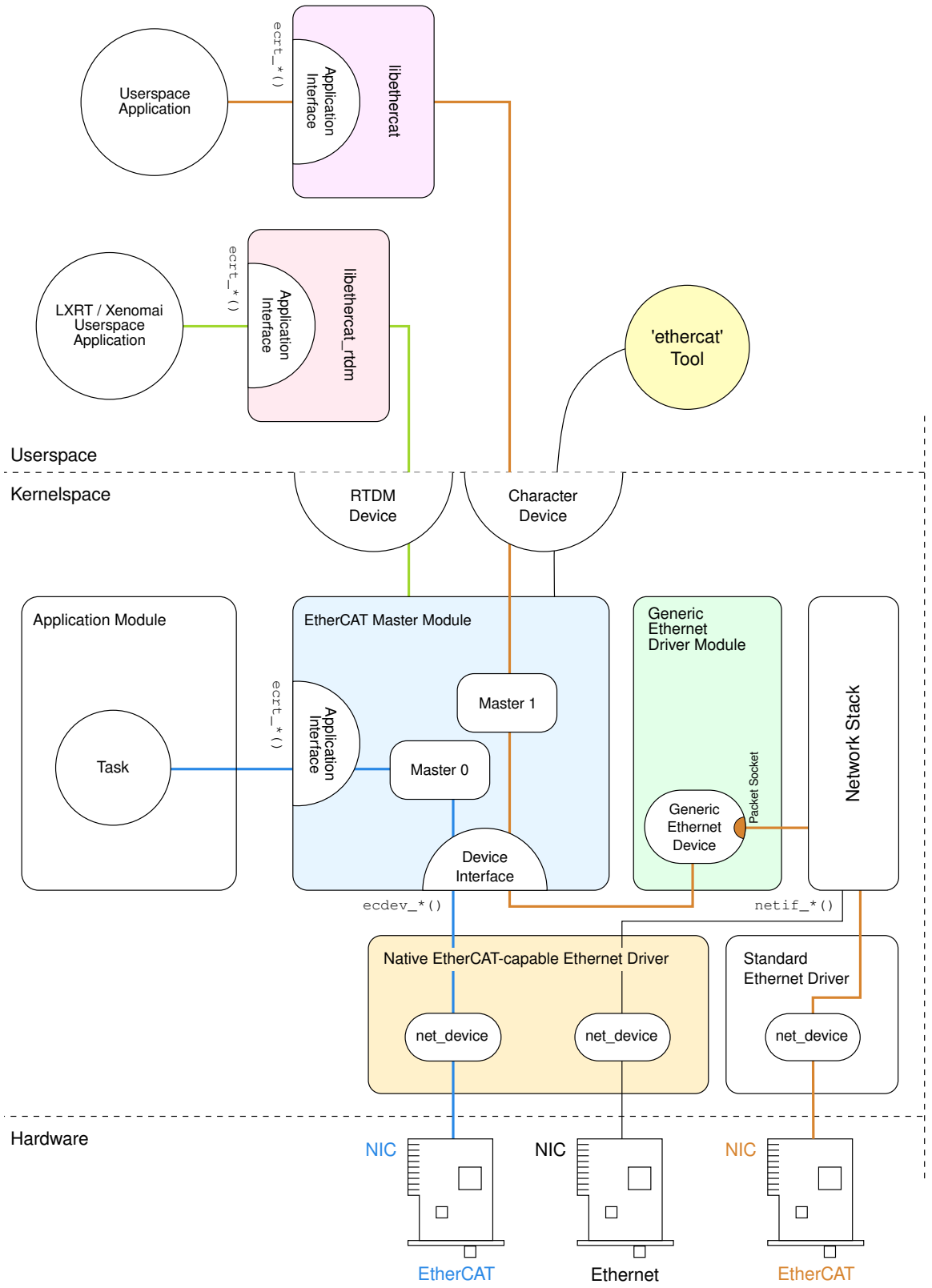

FIGURE 2.1 – Architecture du maître

contrôle du maître : elle peut fournir une configuration de bus et échanger des données de processus. Les applications peuvent être des modules noyaux (qui utilisent directement l'API du noyau) ou des programmes dans l'espace utilisateur, qui utilisent l'API via la bibliothèque EtherCAT (voir section 7.2), ou la bibliothèque RTDM (voir section 7.3).

2.1 Module Maître

Le module noyau du maître EtherCAT *ec_master* peut contenir plusieurs instances maîtresses. Chaque maître attend des périphériques Ethernet particuliers identifiés par leurs adresses MAC. Ces adresses doivent être spécifiées au chargement du module via le paramètre de module *main_devices* (et en option : *backup_devices*). Le nombre d'instances maîtresses à initialiser est défini par le nombre d'adresses MAC fournies.

La commande ci-dessous charge le module maître avec une unique instance maîtresse qui attend un seul périphérique Ethernet dont l'adresse MAC est `00:0E:0C:DA:A2:20`. Le maître sera accessible à l'index 0.

```
# modprobe ec_master main_devices=00:0E:0C:DA:A2:20
```

Pour plusieurs maîtres, des virgules séparent les adresses MAC :

```
# modprobe ec_master main_devices=00:0E:0C:DA:A2:20,00:e0:81:71:d5:1c
```

Les deux maîtres peuvent être adressés par leurs indices respectifs 0 et 1 (voir figure 2.2). L'index du maître est requis par la fonction `ecrt_request_master()` de l'API (voir chapitre 3) et par l'option `--master` de l'outil de commande en ligne *ethercat* (voir section 7.1), qui vaut 0 par défaut.

Niveau de déverminage Le module maître a aussi un paramètre *debug_level* pour configurer le niveau initial de déverminage pour tous les maîtres (voir aussi sous-section 7.1.6).

Script d'initialisation Dans la plupart des cas, il n'est pas nécessaire de charger manuellement le module maître et les modules des pilotes Ethernet. Un script d'initialisation est disponible pour démarrer le maître en tant que service (voir section 7.4). Un fichier de service est aussi disponible pour les systèmes qui sont gérés par systemd [7].

Syslog Le module maître publie des informations à propos de son état et ses événement dans le tampon circulaire du noyau. Elles aboutissent aussi dans les journaux systèmes. La commande de chargement du module devrait produire les messages ci-dessous :

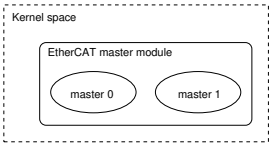

FIGURE 2.2 – Plusieurs maîtres dans un module

```
# dmesg | tail -2
EtherCAT: Master driver 1.5.2
EtherCAT: 2 masters waiting for devices.

# tail -2 /var/log/messages
Jul  4 10:22:45 ethercat kernel: EtherCAT: Master driver 1.5.2
Jul  4 10:22:45 ethercat kernel: EtherCAT: 2 masters waiting
                                 for devices.
```

Les messages du maître sont préfixés par `EtherCAT` pour faciliter la recherche dans les journaux.

2.2 Phases du maître

Chaque maître EtherCAT fourni par le module maître (voir section 2.1) traverse plusieurs phases au cours de son exécution (voir figure 2.3) :

Phase orpheline (Orphaned) Ce mode prend effet quand le maître attend encore pour se connecter à ses périphériques Ethernet. Aucune communication de bus n'est possible pour l'instant.

Phase paresseuse (Idle) Ce mode prend effet quand le maître a accepté tous les périphériques Ethernet requis, mais qu'aucune application ne l'a encore mobilisé. Le maître exécute son automate (voir section 5.3), qui analyse automatiquement le bus pour rechercher les esclaves et exécuter les opérations en attente depuis l'interface en espace utilisateur (par exemple les accès SDO). L'outil en ligne de commande peut être utilisé pour accéder au bus, mais il n'y a aucun échange de donnée de processus parce que la configuration du bus est manquante.

Phase d'opération Le maître est mobilisé par une application qui peut fournir une configuration de bus et échanger des données de processus..

2.3 Données de processus

Cette section présente quelques termes et idées sur la manière dont le maître traite les données de processus.

Image des données de processus Les esclaves présentent leurs entrés et sorties au maître au travers d'objet de données de processus "Process Data Objects" (PDOs). Les PDOs disponibles peuvent être déterminés en lisant les catégories SII TxPDO et RxPDO de l'esclave depuis l'E^2PROM (en cas de PDOs fixes) ou en lisant les objets CoE appropriés (voir section 6.2), si disponibles. L'application peut inscrire les entrées des PDOs pour l'échange pendant l'opération cyclique. La somme de toutes les entrées PDO inscrites définit l'"image des données du processus", qui peut être

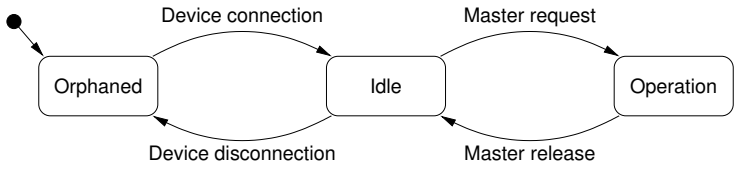

FIGURE 2.3 – Phases et transitions du maître

échangée via des datagrammes avec des accès mémoires “logiques” (comme LWR [2], LRD [3] ou LRW [4]) présentés dans [2, sec. 5.4].

Domaine de données de processus Les images des données de processus peuvent être facilement gérées en créant des “domaines”, qui permettent l'échange de PDO groupés. Ils s'occupent également de gérer les structures des datagrammes qui sont nécessaires pour échanger les PDOs. Les domaines sont obligatoires pour l'échange de données de processus, donc il doit y en avoir au moins un. Ils ont été introduits pour les raisons suivantes :

— La taille maximale d'un datagramme est limitée par celle d'une trame Ethernet. La taille maximale des données est la taille du champ “données” d'Ethernet moins l'entête de la trame Ethernet, moins l'entête du datagramme EtherCAT et moins la terminaison du datagramme EtherCAT : $1500 - 2 - 12 - 2 = 1484$ octets. Si la taille de l'image des données de processus dépasse cette limite, il faut envoyer plusieurs trames et partitionner l'image pour utiliser plusieurs datagrammes. Un domaine gère cela automatiquement.
— Tous les PDOs n'ont pas besoin d'être échangés à la même fréquence : les valeurs des PDOs peuvent varier lentement au cours du temps (par exemple des valeurs de température), aussi les échanger à haute fréquence serait un gaspillage de la bande passante du bus. Pour cette raison, plusieurs domaines peuvent être créés, pour grouper différents PDOs et ainsi séparer les échanges.

Il n'y a aucune limite supérieure pour le nombre de domaines, mais chaque domaine occupe une FMMU [5] dans l'esclave concerné, donc le nombre maximal de domaines est en fait limité par les esclaves.

Configuration FMMU Une application peut inscrire des entrées PDO pour l'échange. Chaque entrée PDO et son PDO parent font partie d'une zone mémoire dans la mémoire physique de l'esclave, qui est protégée par un gestionnaire de synchronisation (sync manager) [2, sec. 6.7] pour des accès synchronisés. Pour que le gestionnaire de synchronisation réagisse à un datagramme qui accède à sa mémoire, il est nécessaire d'accéder au dernier octet couvert par le gestionnaire de synchronisation. Sinon le gestionnaire de synchronisation ne réagira pas au datagramme et aucune donnée ne sera échangée. C'est pourquoi l'ensemble de la zone mémoire synchronisée doit être inclus dans l'image des données de processus : par exemple ; si une entrée PDO particulière d'un esclave est inscrite pour l'échange avec un domaine particulier, une FMMU sera configurée pour mapper toute la mémoire protégée par le gestionnaire de synchronisation dans laquelle l'entrée PDO réside. Si une deuxième entrée PDO du même esclave est inscrite pour l'échange de donnée de processus au sein du même

2. LWR : Logical Write
3. LRD : Logical Read
4. LRW : Logical Read/Write
5. FMMU : Fieldbus Memory Management Unit

domaine, et s'il réside dans la même zone mémoire protégée par le gestionnaire de synchronisation que la première entrée, alors la configuration FMMU n'est pas modifiée, parce que la mémoire désirée fait déjà partie de l'image des données du processus du domaine. Si la deuxième entrée appartenait à une autre zone protégée par le gestionnaire de synchronisation, alors cette zone entière serait aussi incluse dans l'image des données des processus des domaines.

figure 2.4 fournit un aperçu de la manière de configurer les FMMUs pour mapper la mémoire physique vers les images logiques des données des processus.

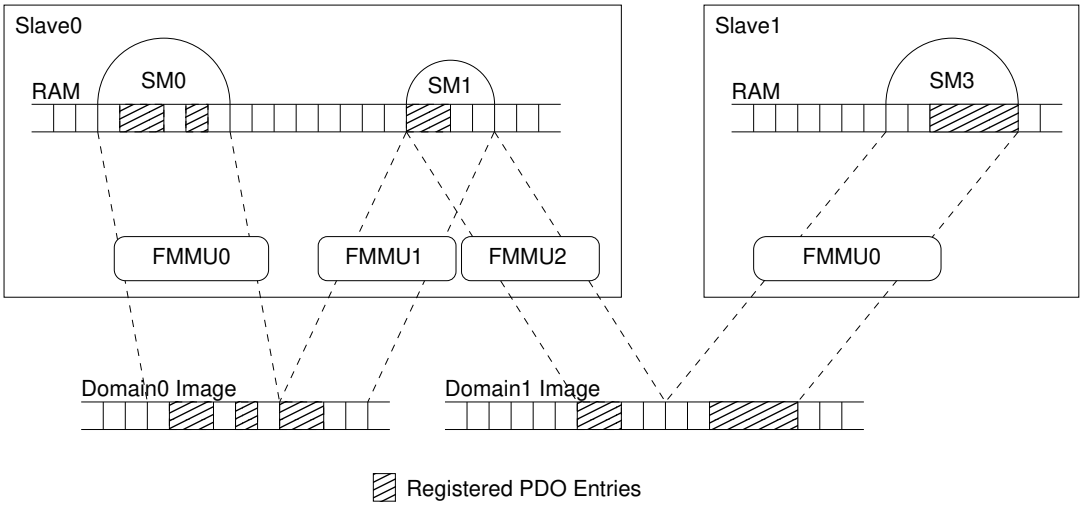

FIGURE 2.4 – Configuration FMMU

3 Interface de Programmation Applicative (API)

L'interface de programmation applicative fournit les fonctions et structures de données pour accéder au maître EtherCAT. La documentation complète de l'interface est incluse sous forme de commentaires Doxygen [13] dans le fichier d'entête *include/ecrt.h*. Elle peut être lue directement depuis les commentaires du fichier, ou plus confortablement sous forme de documentation HTML. La génération du HTML est décrite dans section 9.3.

Les sections suivantes couvrent une description générale de l'API.

Chaque application devrait utiliser le maître en deux étapes :

Configuration Le maître est mobilisé et la configuration est appliquée. Par exemple, les domaines sont créés, les esclaves sont configurés et les entrées PDO sont inscrites. (voir section 3.1).

Opération Le code cyclique est exécuté et les données de processus sont échangées (voir section 3.2).

Exemple d'Applications Il y a quelques exemples d'applications dans le sous-dossier *examples/* du code du maître. Ils sont documentés dans le code source.

3.1 Configuration du maître

La configuration du bus est fournie via l'API. La figure 3.1 donne une vue d'ensemble des objets qui peuvent être configurés par l'application.

3.1.1 Configuration de l'esclave

L'application doit dire au maître quelle est la topologie attendue du bus. Ceci peut être fait en créant des "configurations d'esclaves". Une configuration d'esclave peut être vue comme un esclave attendu. Quand une configuration d'esclave est créée, l'application fournit la position sur le bus (voir ci-dessous), l'identifiant du fabricant (vendor id) et le code du produit (product code).

Quand la configuration du bus est appliquée, le maître vérifie s'il y a un esclave avec l'identifiant du fabricant et le code du produit à la position donnée. Si c'est le

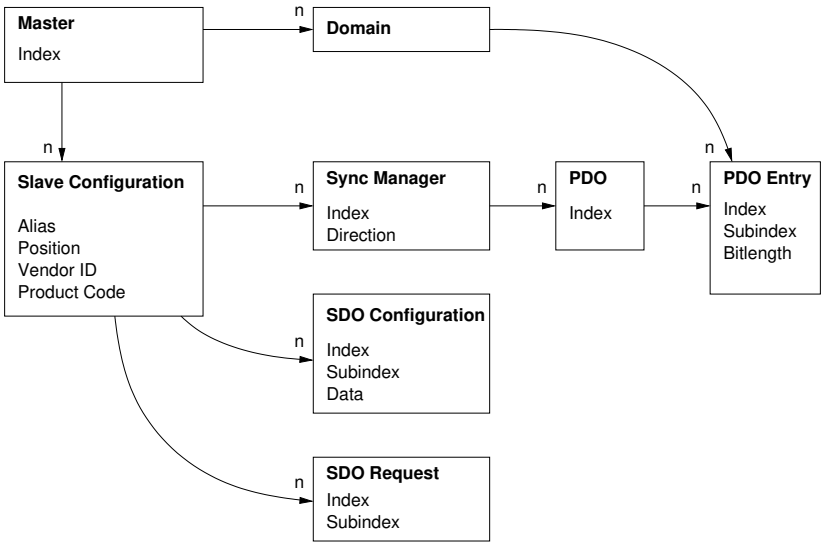

FIGURE 3.1 – Configuration du maître

cas, la configuration de l'esclave est "attachée" à l'esclave réel sur le bus et l'esclave est configuré en fonction des paramètres fournis par l'application. L'état de la configuration de l'esclave peut soit être demandé via l'API ou via l'outil en ligne de commande (voir sous-section 7.1.3).

Position de l'esclave La position de l'esclave doit être spécifiée sous forme d'un couple "alias" et "position". Ceci permet d'adresser les esclaves via la position absolue sur le bus ou via un identifiant stocké et appelé "alias" ou via un mélange des deux. L'alias est une valeur 16 bits stockée dans E^2PROM de l'esclave. Il peut être modifié via l'outil en ligne de commande (voir sous-section 7.1.2). tableau 3.1 montre comment les valeurs sont interprétées.

TABLE 3.1 – Spécifier la position d'un esclave

Alias	Position	Interprétation
`0`	`0` – `65535`	Adressage par position. Le paramètre de position est interprété comme la position absolue de l'anneau sur le bus.
`1` – `65535`	`0` – `65535`	Adressage par alias. Le paramètre de position est interprété comme une position relative après le premier esclave avec une adresse d'alias donnée.

figure 3.2 montre un exemple d'attachement des configurations des esclaves. Certaines configurations sont attachées, tandis que d'autres restes détachées. La liste ci-dessous en donne les raisons en commençant par la configuration de l'esclave du haut.

1. L'alias zéro signifie un adressage simple par position. L'esclave #1 existe et l'identifiant du fabricant et le code produit correspondent aux valeurs attendues.
2. Bien que l'esclave en position 0 a été trouvé, le code produit ne correspond pas, aussi la configuration n'est pas attachée.
3. L'alias n'est pas zéro, aussi l'adressage par alias est utilisé. L'esclave #2 est le premier esclave avec l'alias `0x2000`. Comme la valeur de position est zéro, le même esclave est utilisé.
4. Il n'y a aucun esclave avec l'alias demandé, aussi la configuration ne peut pas être attachée.
5. L'esclave #2 est encore le premier esclave avec l'alias `0x2000`, mais la position est maintenant 1, aussi l'esclave #3 est attaché.

Si les sources du maître sont configurées avec `--enable-wildcards`, alors `0xffffffff` correspond à n'importe quel identifiant de fabricant et/ou code produit.

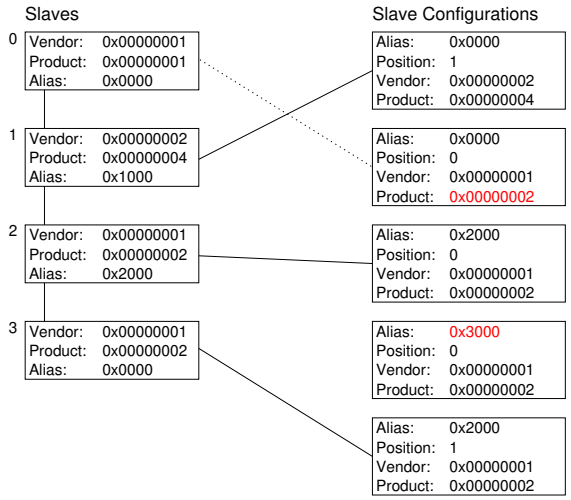

FIGURE 3.2 – Attachement de la configuration des esclaves

3.2 Opération cyclique

Pour entrer dans le mode d'opération cyclique, le maître doit être "activé" pour calculer l'image des données de processus et appliquer la configuration du bus pour la première fois. Après l'activation, l'application est responsable d'envoyer et recevoir les trames. La configuration ne peut pas être modifiée après l'activation.

3.3 Gestionnaires VoE

Pendant la phase de configuration, l'application peut créer des gestionnaires pour le protocole de boîte aux lettres VoE, décrit dans section 6.3. Un gestionnaire VoE appartient toujours à une configuration d'esclave particulière, aussi la fonction de création est une méthode de la configuration de l'esclave.

Un gestionnaire VoE gère les données VoE et les datagrammes utilisés pour transmettre et recevoir les messages VoE. Il contient l'automate nécessaire au transfert des messages VoE.

L'automate VoE peut traiter seulement une opération à la fois. Par conséquent, seule une opération de lecture ou une opération d'écriture peut être émise à un moment donné [1]. Après l'initialisation de l'opération, le gestionnaire doit être exécuté de manière cyclique jusqu'à ce qu'il se termine. Après cela, les résultats de l'opération peuvent être récupérés.

Un gestionnaire VoE a sa propre structure de datagramme, qui est marqué pour l'échange après chaque pas d'exécution. Aussi, l'application peut décider, combien de gestionnaires elle exécute avant d'envoyer les trames EtherCAT correspondantes.

Pour obtenir davantage d'information sur les gestionnaires VoE, consultez la documentation des fonctions de l'API et les exemples d'applications fournis dans le dossier *examples/*.

3.4 Accès concurrents au maître

Dans certains cas, plusieurs instances utilisent un seul maître, par exemple quand une application échange des données de processus cyclique et qu'il y a des esclaves EoE qui ont besoin d'échanger des données Ethernet avec le noyau (voir section 6.1). Pour cette raison, le maître est une ressource partagée qui doit être séquentialisée. Ceci est habituellement réalisé en verrouillant au moyen de sémaphores ou d'autres méthodes pour protéger les sections critiques.

Le maître ne fournit pas lui-même de mécanismes de verrouillage, parce qu'il ne peut connaître le type de verrou approprié. Par exemple, si l'application est en espace noyau et utilise la fonctionnalité RTAI, les sémaphores ordinaires du noyau ne seraient pas

1. Si, on désire envoyer et recevoir simutanément, deux gestionnaires VoE peuvent être créés pour la configuration de l'esclave.

suffisants. Pour cela, une décision de conception importante a été faite : l'application qui a réservé un maître doit en avoir le contrôle total, c'est pourquoi elle doit prendre la responsabilité de fournir les mécanismes de verrouillage appropriés. Si une autre instance veut accéder au maître, elle doit demander l'accès au bus via des fonctions de rappels qui doivent être fournis par l'application. De plus, l'application peut refuser l'accès au maître, si elle considère que le moment est gênant.

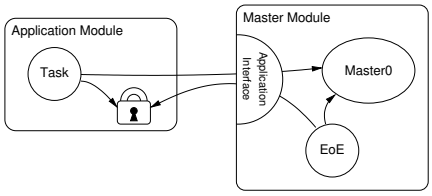

FIGURE 3.3 – Accès concurrent au maître

L'exemple figure 3.3 montre comment deux processus partagent un maître : la tâche cyclique de l'application utilise le maître pour l'échange de données de processus, tandis que le processus EoE interne au maître l'utilise pour communiquer avec les esclaves EoE. Les deux ont accès au bus de temps en temps, mais le processus EoE le fait en "demandant" à l'application de réaliser l'accès au bus pour lui. De cette manière, l'application peut utiliser le mécanisme de verrouillage approprié pour éviter d'accèder au bus en même temps. Voir la documentation de l'API (chapitre 3) pour savoir comment utiliser ces fonctions de rappel.

3.5 Horloges distribuées

À partir de la version 1.5, le maître supporte les “horloges distribuées” (Distributed Clocks) EtherCAT pour synchroniser les horloges des esclaves sur le bus avec l’horloge de “référence” (qui est l’horloge locale du premier esclave qui supporte l’horloge distribuée) et pour synchroniser l’horloge de référence avec “l’horloge maîtresse” (qui est l’horloge locale du maître). Toutes les autres horloges du bus (après l’horloge de référence) sont considérés comme “horloges esclaves” (voir figure 3.4).

Horloges locales Tout esclave EtherCAT qui supporte l’horloge distribuée possède un registre d’horloge locale avec une résolution à la nanoseconde. Si l’esclave est allumé, l’horloge démarre depuis zéro, ce qui signifie que lorsque des esclaves sont allumés à différents instants, leurs horloges auront des valeurs différentes. Ces “décalages” doivent être compensés par le mécanisme des horloges distribuées. En outre, les horloges ne tournent pas exactement à la même vitesse, puisque les quartzs ont une déviation de leur fréquence naturelle. Cette déviation est habituellement très faible, mais au bout de longues périodes, l’erreur s’accumulera et la différence entre les horloges locales grandira. Cette “dérive” des horloges doit aussi être compensée par le mécanisme des horloges distribuées.

Temps de l’Application La base de temps commune pour le bus doit être fournie par l’application. Ce temps d’application t_{app} est utilisé

1. pour configurer les décalages des horloges des esclaves (voir ci-dessous),
2. pour programmer les temps de démarrage de l’esclave pour la génération des impulsions synchrones. (voir ci-dessous)
3. pour synchroniser les horloges de référence avec l’horloge maîtresse (optionnel).

Compensation du décalage Pour la compensation du décalage, chaque esclave fournit un registre de “décalage du temps système” t_{off}, qui est ajouté à la valeur de l’horloge interne t_{int} pour obtenir le “Temps Système” t_{sys} :

$$\begin{aligned} t_{\text{sys}} &= t_{\text{int}} + t_{\text{off}} \\ \Rightarrow t_{\text{int}} &= t_{\text{sys}} - t_{\text{off}} \end{aligned} \tag{3.1}$$

Le maître lit les valeurs des deux registres pour calculer un nouveau décalage du temps système de telle manière que le temps système résultant corresponde au temps de l’application du maître t_{app} :

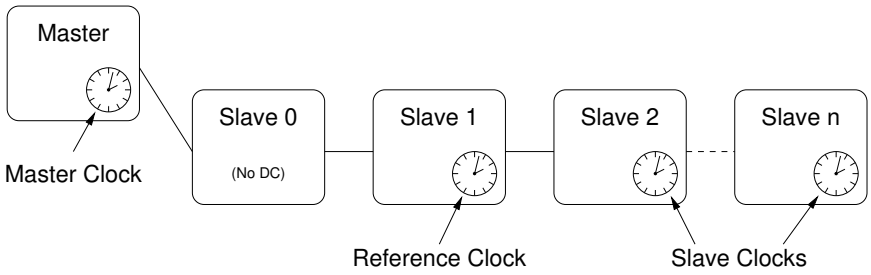

FIGURE 3.4 – Horloges distribuées

$$t_{\text{sys}} \stackrel{!}{=} t_{\text{app}} \tag{3.2}$$

$$\begin{aligned} \Rightarrow t_{\text{int}} + t_{\text{off}} &\stackrel{!}{=} t_{\text{app}} \\ \Rightarrow t_{\text{off}} &= t_{\text{app}} - t_{\text{int}} \\ \Rightarrow t_{\text{off}} &= t_{\text{app}} - (t_{\text{sys}} - t_{\text{off}}) \\ \Rightarrow t_{\text{off}} &= t_{\text{app}} - t_{\text{sys}} + t_{\text{off}} \end{aligned} \tag{3.3}$$

La petite erreur de décalage du temps résultant des différences de temps entre la lecture et l'écriture des registres sera compensée par la compensation de la dérive.

Compensation de la dérive La compensation de la dérive est possible grâce à un mécanisme spécial de chaque esclave compatible avec les horloges distribuées : une opération d'écriture dans le registre du "Temps système" obligera la boucle de contrôle du temps interne à comparer le temps écrit (moins le délai de transmission programmé, voir ci-dessous) avec le temps système courant. L'erreur de temps calculée sera utilisée comme une entrée pour le contrôleur de temps, qui ajustera la vitesse de l'horloge locale pour être légèrement plus rapide ou plus lente [2], en fonction du signe de l'erreur.

Délais de transmission La trame Ethernet a besoin d'une petite quantité de temps pour se propager d'esclave en esclave. Les délais de transmission s'accumulent sur le bus et peuvent attendre la magnitude de la microseconde et doivent alors être pris en compte par la compensation de la dérive. Les esclaves EtherCAT qui supportent les horloges distribuées fournissent un mécanisme pour mesurer les délais de transmission : pour chacun des 4 ports de l'esclave il y a un registre d'heure de réception. Une opération d'écriture sur le registre d'heure de réception du port démarre la mesure et l'heure système courante est capturée et stockée dans un registre d'heure de réception une fois que la trame est reçue sur le port correspondant. Le maître peut lire le temps de réception relatif puis calculer les délais entre les esclaves (en utilisant sa connaissance de la topologie du bus), et finalement calculer les délais de chaque esclave avec l'horloge de référence. Ces valeurs sont programmées dans les registres de délai de transmission des esclaves. De cette manière, la compensation de la dérive peut attendre une synchronie à la nanoseconde.

Vérification de la synchronie Les esclaves compatibles avec les horloge distribuées fournissent un registre 32 bits "Différence de l'heure système" à l'adresse `0x092c`, dans lequel la différence de temps système de la dernière compensation de la dérive est stockée avec une résolution d'une nanoseconde et un codage signe-et-magnitude [3]. Pour vérifier la synchronie du bus, les registres de différence du temps système peuvent aussi être lus via l'outil en ligne de commande (voir sous-section 7.1.14) :

2. L'horloge locale de l'esclave sera incrémentée de 9 ns, 10 ns ou 11 ns toute les 10 ns.
3. Ceci permet une lecture-diffusion de tous les registres de différence de temps système sur le bus pour obtenir une approximation de la valeur supérieure.

```
$ watch -n0 "ethercat reg_read -p4 -tsm32 0x92c"
```

Signaux synchrones Les horloge synchrones sont seulement un pré-requis pour des évènements synchrones sur le bus. Chaque esclave qui supporte les horloges distribuées fournit deux “signaux synchrones”, qui peuvent être programmés pour créer des évènements, qui vont par exemple obliger l’application esclave à capturer ses entrées à un instant précis. Un évènement synchrone peut être généré soit une seule fois ou périodiquement, selon ce qui a du sens pour l’application esclave. La programmation des signaux synchrones est une question de réglage du mot “AssignActivate” et des temps de cycle et décalage des signaux de synchronisation. Le mot AssignActivate est spécifique à chaque esclave et doit être récupéré depuis la description XML de l’esclave (`Device` → `Dc`), où se trouvent aussi typiquement les signaux de configurations “OpModes”.

4 Interfaces Ethernet

Le protocole EtherCAT est fondé sur le standard Ethernet standard, aussi un maître dépend du matériel Ethernet standard pour communiquer avec le bus.

Le terme *device* est utilisé comme synonyme pour matériel d'interface réseau Ethernet.

Pilotes natifs pour périphériques Ethernet Il y a des modules natifs pour les pilotes de périphériques (voir section 4.2) qui gèrent le matériel Ethernet qu'utilise le maître pour se connecter au bus EtherCAT. Ils offrent leurs matériels Ethernet au module maître via l'interface de device (voir section 4.6) et doivent être capable de préparer les périphériques Ethernet pour les opérations EtherCAT (temps réel) ou pour les opérations "normales" en utilisant la pile réseau du noyau. L'avantage de cete approche est que le maître peut opérer pratiquement directement avec le matériel ce qui permet des performances élevées. L'inconvénient est qu'il faut avoir une version compatible EtherCAT du pilote Ethernet original.

Pilote générique pour les périphériques Ethernet À partir du maître version 1.5, il y a un module de pilote générique pour les périphériques Ethernet (voir section 4.3), qui utilise les couches basses de la pile réseau pour se connecter au matériel. L'avantage est que n'importe quel périphérique Ethernet peut être utilisé pour les opérations EtherCAT, indépendamment du pilote matériel réel (ainsi tous les pilotes Ethernet Linux sont supportés sans modification). L'inconvénient est que cette approche ne supporte pas les extensions temps réel, comme RTAI, parce que la pile réseau de Linux est utilisée. Cependant la performance est légèrement moins bonne qu'avec l'approche native, car les données de la trame Ethernet doivent traverser la pile réseau.

4.1 Principes de base du pilote réseau

EtherCAT repose sur le matériel Ethernet et le maître a besoin d'un périphérique Ethernet physique pour communiquer avec le bus. C'est pourquoi, il est nécessaire de comprendre comment Linux gère les périphériques réseaux et leurs pilotes.

Tâches d'un pilote réseau Les pilotes de périphériques réseaux gèrent habituellement les deux couches les plus basses du modèle OSI, qui sont la couche physique et la couche liaison de données. Le périphérique réseau gère nativement les problèmes de la

couche physique : il représente le matériel pour se connecter au média et pour envoyer et recevoir des données de la manière décrite par le protocole de la couche physique. Le pilote de périphérique réseau est responsable de récupérer les données depuis la pile réseau du noyau et de les faire suivre au périphérique qui fait la transmission physique. Si des données sont reçues par le périphérique alors le pilote est notifié (habituellement au moyen d'une interruption) et il doit lire les données depuis la mémoire du périphérique et l'envoyer à la pile réseau. Un pilote de périphérique réseau doit aussi gérer d'autres tâches telles que le contrôle de la file d'attente, les statistiques et les fonctionnalités spécifiques du périphérique.

Démarrage du pilote Habituellement, un pilote recherche des périphériques compatibles lors du chargement du module. Pour les pilotes PCI, ceci est fait en analysant le bus PCI et en vérifiant les identifiants (ID) des périphériques. Si un périphérique est trouvé, les structures de données sont allouées et le périphérique est mis en service.

Fonctionnement des interruptions Un périphérique réseau fournit généralement une interruption matérielle qui est utilisée pour notifier le pilote des trames reçues et des succès ou erreurs des transmissions. Le pilote doit enregistrer une routine de service d'interruption – en anglais *interrupt service routine* – (ISR), qui est exécutée à chaque fois que le matériel signale un tel évènement. Si l'interruption a été envoyée par le bon périphérique (plusieurs périphériques peuvent partager une même interruption matérielle), la raison de l'interruption doit être déterminée en lisant le registre d'interruption du périphérique. Par exemple, si le drapeau pour les trames reçues est activé, les données des trames doivent être copiées depuis le matériel vers la mémoire du noyau puis transmise à la pile réseau.

La structure `net_device` Le pilote enregistre une structure `net_device` pour chaque périphérique pour communiquer avec la pile réseau et créé une "interface réseau". Dans le cas d'un pilote Ethernet, cette interface apparaît sous la forme *ethX*, où X est le numéro assigné par le noyau à l'enregistrement. La structure `net_device` reçoit les évènements (soit depuis l'espace utilisateur, soit depuis la pile réseau) via différentes fonctions de rappel, qui doivent être définies avant l'enregistrement. Toutes les fonctions de rappel ne sont pas obligatoires, mais pour un fonctionnement raisonnable, celles qui sont définies ci-dessous sont nécessaires dans tous les cas :

`open()` Cette fonction est appelée quand la communication a démaré, par exemple après une commande `ip link set ethX up` depuis l'espace utilisateur. La réception des trames doit être activée par le pilote.

`stop()` Le but de cette fonction de rappel est de "fermer" le périphérique, c'est-à-dire faire en sorte que le matériel cesse de recevoir des trames.

`hard_start_xmit()` Cette fonction est appelée pour chaque trame qui a été transmise. La pile réseau passe la trame sous la forme d'un pointeur vers une structure

`sk_buff` (“socket buffer” – tampon de socket – voir ci-dessous), qui doit être libérée après l'envoi.

`get_stats()` Cet appel doit retourner un pointeur vers la structure `net_device_stats`, qui doit être continuellement mise à jour avec les statistiques des trames. Cela signifie qu'à chaque fois qu'une trame est reçue, envoyée ou qu'une erreur se produit, le compteur approprié de cette structure doit être augmenté.

L'inscription réelle est faite par l'appel `register_netdev()`, la désinscription est faite par `unregister_netdev()`.

L'interface `netif` Toute autre communication dans la direction interface → réseau est faite via les appels `netif_*()`. Par exemple, après l'ouverture réussie du périphérique, la pile réseau doit être notifiée, pour qu'elle puisse maintenant passer les trames à l'interface. Ceci est fait en appelant `netif_start_queue()`. Après cet appel, la fonction de rappel `hard_start_xmit()` peut être rappelée par la pile réseau. De plus, un pilote réseau gère habituellement une file d'attente pour la transmission des trames. Quand elle est pleine, il faut informer la pile réseau qu'elle doit cesser de pousser davantage de trames pendant un moment. Ceci se produit avec un appel à `netif_stop_queue()`. Si des trames ont été envoyées, et qu'il y a à nouveau suffisamment de place pour les mettre en file d'attente, ceci peut être notifié avec `netif_wake_queue()`. Un autre appel important est `netif_receive_skb()` [1] : il passe une trame qui vient juste d'être reçue par le périphérique, à la pile réseau. Les données de la trame doivent être incluses à cet effet dans le “tampon de socket” (voir ci-dessous).

Tampons de Socket Les tampons de sockets sont le type de données fondamental de toute la pile réseau. Ils servent de container pour les données réseaux et sont capables d'ajouter rapidement des données au début et à la fin, ou bien de les retirer. C'est pourquoi, un tampon de socket consiste en un tampon alloué et plusieurs pointeurs qui marquent le début du tampon (`head`), le début des données data (`data`), la fin des données (`tail`) et la fin du tampon (`end`). De plus, un tampon de socket contient les informations d'entête pour le réseau et (en cas de données reçue), un pointeur vers le `net_device`, qui l'a réceptionné. Il existe des fonctions qui créent un tampon socket (`dev_alloc_skb()`), ajoutent des données au début (`skb_push()`) ou à la fin (`skb_put()`), suppriment des données au début (`skb_pull()`) ou à la fin (`skb_trim()`), ou suppriment le tampon (`kfree_skb()`). Un tampon socket est passé de couche en couche et il est libéré par la couche qui s'en sert en dernier. En cas d'envoi, la libération est faite par le pilote réseau.

1. Cette fonction fait partie de NAPI (“New API”), qui remplace la technique du noyau 2.4 pour interfacer la pile réseau (avec `netif_rx()`). NAPI est une technique pour améliorer la performance réseau de Linux. Davantage d'information dans http://www.cyberus.ca/~hadi/usenix-paper.tgz.

4.2 Les pilotes natifs pour périphériques EtherCAT

Il y a quelques conditions qui s'appliquent au matériel Ethernet lorsqu'il est utilisé avec un pilote Ethernet natif avec les fonctionnalités EtherCAT.

Matériel dédié Pour des raisons de performances et de temps réel, le maître EtherCAT a besoin d'un accès direct et exclusif au matériel Ethernet. Cela implique que le périphérique réseau ne doit pas être connecté à la pile réseau du noyau comme d'habitude, car le noyau essaierait de l'utiliser comme un périphérique Ethernet ordinaire.

Opération sans interruption Les trames EtherCAT voyagent au travers de l'anneau logique EtherCAT et sont alors renvoyées au maître. La communication est hautement déterministe : une trame est envoyée et sera reçue après un temps constant, aussi il n'y pas besoin de notifier le pilote de la réception de la trame. À la place, le maître peut interroger le matériel pour les trames reçues, s'il s'attend à ce qu'elles soient déjà arrivées.

La figure 4.1 montre deux flots de travail pour la transmission et réception cyclique de trames avec et sans interruptions.

Dans le flux de travail de gauche, "Opération avec interruption", les données venant du dernier cycle sont d'abord traitées et une nouvelle trame est assemblée avec des nouveaux datagrammes, puis elle est envoyée. Le travail cyclique est fait pout l'instant. Plus tard, quand la trame est à nouveau reçue par le matériel, une interruption est déclenchée et l'ISR est exécutée. L'ISR va récupérer les données de la trame depuis le matériel et commencer la dissection de la trame : les datagrammes seront traités, et alors les données seront prêtes pour le traitement dans le prochain cycle.

Dans le flux de travail de droite, "Opération sans interruption", aucune interruption matérielle n'est activée. À la place, le maître va sonder le matériel en exécutant l'ISR. Si la trame a été reçue entre temps, elle sera disséquée. La situation est maintenant la même qu'au début de flux de travail de gauche : les données reçues sont traitées et une nouvelle trame est assemblée et envoyée. Il n'y a rien d'autre à faire pour le reste du cycle.

L'opération sans interruption est préférable, parce que les interruptions matérielles ne sont pas propices à l'amélioration du comportement temps réel du pilote : leurs incidences indéterministes contribuent à augmenter la gigue. En outre, si une extension temps réel (comme RTAI) est utilisée, un effort supplémentaire devra être fait pour hiérarchiser les interruptions.

Périphériques Ethernet et EtherCAT Un autre problème réside dans la façon dont Linux gère les périphériques du même type. Par exemple, un pilote PCI analyse le bus PCI pour chercher des périphériques qu'il peut gérer. Alors, il s'enregistre lui-même comme pilote responsable pour tous les périphériques trouvés. Le problème est que

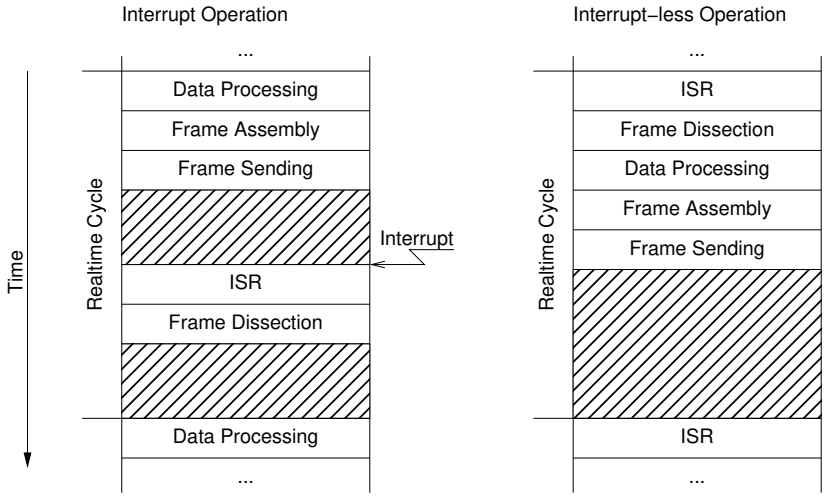

FIGURE 4.1 – Opération avec interruption versus Opération sans interruption

l'on ne peut pas dire à un pilote non modifié d'ignorer un périphérique pour l'utiliser ultérieurement pour EtherCAT. Il faut donc un moyen de gérer plusieurs périphériques du même type, l'un étant réservé à EtherCAT, tandis que l'autre est traité comme un périphérique Ethernet ordinaire.

Pour toutes ces raisons, l'auteur a décidé que la seule solution acceptable était de modifier les pilotes Ethernet standards de manière à ce qu'ils conservent leurs fonctionnalités normales, tout en gagnant la possibilité de traiter un ou plusieurs périphériques comme étant compatibles EtherCAT.

Les avantages de cette solution sont listés ci-dessous :

— Pas besoin de dire aux pilotes standards d'ignorer certains périphériques.
— Un seul pilote réseau pour les périphériques EtherCAT et non-EtherCAT.
— Pas besoin d'implémenter un pilote réseau depuis zéro et de rencontrer des problèmes que les anciens développeurs ont déjà résolus.

L'approche choisie a les inconvénients suivants :

— Le pilote modifié est plus compliqué car il doit gérer les périphériques EtherCAT et non-EtherCAT.
— De nombreuses différenciations de cas supplémentaires dans le code du pilote.
— Les modifications et changements dans les pilotes standards doivent être portés de temps en temps vers les versions compatibles EtherCAT.

4.3 Le pilote de périphérique EtherCAT générique

Puisqu'il existe des approches pour activer un fonctionnement en temps réel [12] du noyau Linux complet, il est possible d'opérer sans implémentation native des pilotes de périphériques Ethernet compatibles EtherCAT et d'utiliser la pile réseau à la place. La figure 2.1 présente le "Module de pilote Ethernet générique", qui se connecte à des périphériques Ethernet locaux via la pile réseau. Le module noyau se nomme `ec_generic` et il peut être chargé après le module maître comme un pilote Ethernet compatible EtherCAT.

Le pilote de périphérique générique analyse la pile réseau à la recherche d'interfaces enregistrées par les pilotes de périphériques Ethernet. Il offre tous les périphériques possibles au maître EtherCAT. Si le maître accepte un périphérique, le pilote générique crée un socket de paquet (voir `man 7 packet`) avec `socket_type` mis à `SOCK_RAW`, lié à ce périphérique. Toutes les fonctions de l'interface de ce périphérique (voir section 4.6) opéreront alors sur ce socket.

Les avantages de cette solution sont listés ci-dessous :

— Tout matériel, qui est géré par un pilote Ethernet Linux, peut être utilisé pour EtherCAT.
— Aucune modification n'est nécéssaire sur les pilotes Ethernet réels.

L'approche générique a les inconvénients suivants :

— La performance est un peut moins bonne qu'avec l'approche native, parce que les données de la trame doivent traverser les couches basses de la pile réseau.
— Il n'est pas possible d'utiliser des extensions en temps réel dans le noyau comme RTAI avec le pilote générique, car le code de la pile réseau utilise des allocations dynamiques de mémoire et d'autres choses, qui pourraient provoquer le gel du système dans un contexte temps réel.

Activation du périphérique Dans le but d'envoyer et recevoir des trames au travers d'un socket, le périphérique Ethernet lié à ce socket doit être activé, autrement toutes les trames seront rejetées. L'activation doit avoir lieu avant le chargement du module maître et peut avoir lieu de différentes manières :

— Ad-hoc, en utilisant la commande `ip link set dev ethX up` (ou la commande plus ancienne `ifconfig ethX up`),
— Configurée, en fonction de la distribution, par exemple en utilisant les fichiers `ifcfg` (`/etc/sysconfig/network/ifcfg-ethX`) dans openSUSE et d'autres. C'est le meilleur choix si le maître EtherCAT doit démarrer avec le système. Puisque le périphérique Ethernet doit seulement être activé, mais qu'aucune adresse IP etc. ne sera assignée, il est suffisant d'utiliser `STARTMODE=auto` comme configuration.

4.4 Fourniture de périphériques Ethernet

Après le chargement du module maître, des modules additionnels doivent être chargés pour offrir des périphériques au(x) maître(s) (voir section 4.6). Le module maître connaît les périphériques à choisir grâce aux paramètres de module (voir section 2.1). Si le script d'initialisation est utilisé pour démarrer le maître, les pilotes et périphériques à utiliser peuvent être spécifiés dans le fichier sysconfig (voir sous-section 7.4.2).

Les modules offrant des périphériques Ethernet peuvent être

— des modules natifs de pilotes réseaux compatibles EtherCAT (voir section 4.2) ou
— le module générique de périphérique EtherCAT (voir section 4.3).

4.5 Redondance

L'opération redondante de bus signifie, qu'il y a plus qu'une connexion Ethernet entre le maître et les esclaves. Les datagrammes de l'échange de données de processus sont envoyés sur chaque lien maître, aussi l'échange se terminera, même si le bus est déconnecté quelque part entre les deux.

La condition pour une opération redondante de bus est que chaque esclave puisse être atteint par au moins un lien maître. Dans ce cas, une panne de connexion unique

(i. e. la rupture d'un câble) ne conduira jamais à des données de processus incomplètes. Les doubles défauts ne peuvent pas être traités avec deux périphériques Ethernet.

La redondance peut être configurée avec le commutateur `--with-devices` au moment de la configuration (voir chapitre 9) et en utilisant le paramètre `backup_devices` du module noyau `ec_master` (voir section 2.1) ou la variable appropriée `MASTERx_BACKUP` dans le fichier de configuration sysconfig (voir sous-section 7.4.2).

L'analyse du bus est faite après un changement de topologie sur n'importe quel lien Ethernet. L'API (voir chapitre 3) et l'outil en ligne de commande (voir section 7.1) ont tous les deux des méthodes pour interroger le status de l'opération redondante.

4.6 Interface de périphérique EtherCAT

Une anticipation de la section concernant le module maître (section 2.1) est nécessaire pour comprendre la manière dont un module de pilote de périphérique réseau peut connecter un périphérique à un maître EtherCAT spécifique.

Le module maître fournit une "interface de périphérique" pour les pilotes de périphériques réseaux. Pour utiliser cette interface, un module de pilote de périphérique réseau doit inclure l'entête *devices/ecdev.h*, provenant du code du maître EtherCAT. Cet entête offre une interface de fonction pour les périphériques EtherCAT. Toutes les fonctions de l'interface du périphérique sont nommées avec le préfixe `ecdev`.

La documentation de l'interface du périphérique peut être trouvée dans le fichier d'entête ou dans le module approprié de la documentation de l'interface (voir section 9.3 pour les instruction pour la générer).

4.7 Application de correctifs aux pilotes de réseau natifs

Cette section décrit, comment fabriquer un pilote Ethernet standard compatible EtherCAT, en utilisant l'approche native (voir section 4.2). Malheureusement, il n'y a pas de procédure standard pour permettre l'utilisation d'un pilote Ethernet par le maître EtherCAT, mais il existe quelques techniques courantes.

1. Une première règle simple est d'éviter les appels `netif_*()` pour tous les périphériques EtherCAT. Comme indiqué précédemment, les périphériques EtherCAT ne doivent avoir aucune connexion avec la pile réseau, et c'est pourquoi ils ne doivent pas appeler ces fonctions d'interface.
2. Une autre chose importante est, que les périphériques EtherCAT doivent fonctionner sans interruption. Aussi tous les appels pour inscrire les gestionnaires d'interruption et activer les interruptions au niveau matériel doivent aussi être évités.

3. Le maître n'utilise pas un nouveau tampon de socket pour chaque opération d'envoi : à la place, il y a un tampon fixe, alloué pendant l'initialisation du maître. Ce tampon de socket est rempli avec une trame EtherCAT par chaque opération d'envoi et transmis à la fonction de rappel `hard_start_xmit()`. C'est pourquoi, il est nécessaire que le tampon de socket ne soit pas libéré comme d'habitude par le pilote réseau.

Un pilote Ethernet gère habituellement plusieurs périphériques Ethernet, chacun est décrit par une structure `net_device` avec un champ `priv_data` pour attacher les données qui dépendent du pilote à la structure. Pour distinguer entre les périphériques Ethernet normaux et ceux qui sont utilisés par les maîtres EtherCAT, la structure de données privées utilisée par le pilote peut être étendue avec un pointeur, qui pointe vers un objet `ec_device_t` retourné par `ecdev_offer()` (voir section 4.6) si le périphérique est utilisé par un maître ou sinon qui est à zéro.

Le pilote Ethernet RealTek RTL-8139 est un pilote Ethernet "simple" qui peut servir d'exemple pour modifier des nouveaux pilotes. Les sections intéressantes peuvent être trouvées en recherchant la chaîne "ecdev" dans le fichier *devices/8139too-2.6.24-ethercat.c*.

5 Automates finis

Beaucoup de parties du maître EtherCAT sont implémentées sous forme d' *automates finis* – en anglais *finite state machines* (FSMs). Bien qu'ils amènent une plus grande complexité pour certains aspects, ils ouvrent de nombreuses nouvelles possibilités.

Le court exemple de code ci-dessous montre comment lire tous les états d'esclave et illustre en outre les restrictions du codage " séquentiel " :

```
1 ec_datagram_brd(datagram, 0x0130, 2); // prepare datagram
2 if (ec_master_simple_io(master, datagram)) return -1;
3 slave_states = EC_READ_U8(datagram->data); // process datagram
```

La fonction *ec_master_simple_io()* fournit une interface simple pour envoyer de manière synchrone un datagramme unique et recevoir le résultat [1]. En interne, elle met en file d'attente le datagramme spécifié, invoque la fonction *ec_master_send_datagrams()* pour envoyer une trame avec le datagramme en attente, puis attend activement la réception.

Cette approche séquentielle est très simple, se reflétant dans seulement trois lignes de code. L'inconvénient est que le maître est bloqué pendant le temps où il attend la réception du datagramme. Ce n'est pas vraiment un problème, s'il n'y a qu'une seule instance qui utilise le maître, mais si plusieurs instances veulent (de manière synchrone [2]) utiliser le maître, il est inévitable de songer à une alternative au modèle séquentiel.

L'accès maître doit être séquentalisé pour que plusieurs instances puissent envoyer et recevoir des datagrammes de manière synchrone. Avec la présente approche, cela se traduirait par une phase d'attente active pour chaque instance, ce qui serait inacceptable, en particulier dans des circonstances en temps réel, en raison de l'énorme surcharge de temps.

Une solution possible serait, que toutes les instances soient exécutées séquentiellement pour mettre en file d'attente leurs datagrammes, et qu'elles passent alors le contrôle à la prochaine instance au lieu d'attendre la réception du datagramme. Finalement, une instance supérieure ferait l'entrée-sortie sur le bus pour envoyer et recevoir tous

1. Comme tous les problèmes de communication ont été entre temps transmis aux automates finis, la fonction est obsolète et a cessé d'exister. Néanmoins, elle est suffisante pour montrer ses propres restrictions.

2. À ce stade, l'accès synchrone au maître sera suffisant pour montrer les avantages d'un automate. L'approche asynchrone sera discutée dans la section 6.1

les datagrammes en attente. La prochaine étape serait d'exécuter à nouveau toutes les instances pour qu'elles traitent leurs datagrammes reçus et en émettent des nouveaux.

Cette approche aboutit à ce que toutes les instances mémorisent leurs états lorsqu'elles redonnent le contrôle à l'instance supérieure. Il est évident dans ce cas d'utiliser le modèle d'*automate*. La section 5.1 introduira une partie de la théorie utilisée, tandis que l'extrait ci-dessous montre l'approche de base en codant l'exemple ci-dessus sous forme d'automate :

```
// state 1
ec_datagram_brd(datagram, 0x0130, 2); // prepare datagram
ec_master_queue(master, datagram); // queue datagram
next_state = state_2;
// state processing finished
```

Après que toutes les instances ont exécuté leur état courant et mis en file d'attente leurs datagrammes, ceci sont envoyés et reçus. Alors les états suivants respectifs sont exécutés :

```
// state 2
if (datagram->state != EC_DGRAM_STATE_RECEIVED) {
        next_state = state_error;
        return; // state processing finished
}
slave_states = EC_READ_U8(datagram->data); // process datagram
// state processing finished.
```

Voir section 5.2 pour une introduction au concept de programmation d'automate fini utilisé dans le code du maître.

5.1 Théorie des automates finis

Un automate fini [9] est un modèle de comportement avec des entrées et des sorties, où les sorties dépendent non-seulement des entrées, mais aussi de l'historique des entrées. La définition mathématique d'un automate fini (ou automate avec un nombre fini d'états) est un six-tuple $(\Sigma, \Gamma, S, s_0, \delta, \omega)$, avec

- l'alphabet d'entrée Σ, avec $\Sigma \neq \emptyset$, contenant tous les symboles d'entrées,
- l'alphabet de sortie Γ, avec $\Gamma \neq \emptyset$, contenant tous les symboles de sorties,
- l'ensemble des états S, avec $S \neq \emptyset$,
- l'ensemble des états initiauxs s_0 avec $s_0 \subseteq S, s_0 \neq \emptyset$
- la fonction de transition $\delta : S \times \Sigma \to S \times \Gamma$
- la fonction de sortie ω.

La fonction de transition d'état δ est souvent spécifiée sous la forme d'une *table de transition d'état*, ou par un *diagramme de transition d'état*. La table de transition offre une vue matricielle du comportement de l'automate fini (voir tableau 5.1). Les lignes

de la matrice correspondent aux états ($S = \{s_0, s_1, s_2\}$) et les colonnes correspondent aux symboles d'entrée ($\Gamma = \{a, b, \varepsilon\}$). Le contenu de la table à la ligne i et à la colonne j représente alors le prochain état (et éventuellement la sortie) pour le cas où le symbole σ_j est lu dans l'état s_i.

TABLE 5.1 – Une table typique de transition d'état

	a	b	ε
s_0	s_1	s_1	s_2
s_1	s_2	s_1	s_0
s_2	s_0	s_0	s_0

Le diagramme d'état pour le même exemple est semblable à figure 5.1. Les états sont représentés par des cercles ou des ellipses et les transitions sont représentées par des flèches entre eux. La condition à remplir pour autoriser la transition se trouve à proximité de la flèche de transition. L'état initial est marqué par un disque noir avec une flèche pointant vers l'état respectif.

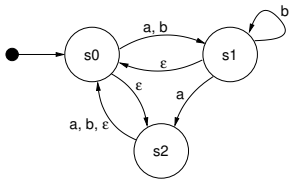

FIGURE 5.1 – Un diagramme typique de transition d'état

Automate fini déterministe et non-déterministe Un automate fini peut être déterministe, ce qui signifit que pour un état et une entrée, il y a un (et seulement un) état suivant. Dans ce cas, l'automate fini a exactement un état de départ. Les automates finis non-déterministes peuvent avoir plusieurs transitions pour une paire unique état-entrée. Il existe un ensemble d'états de départ dans ce dernier cas.

Automates de Moore et de Mealy Il y a une distinction entre ce qu'on appelle les *automates de Moore*, et les *automates de Mealy*. Mathématiquement parlant, la distinction se situe dans la fonction de sortie ω : si elle ne dépend que de l'état courant ($\omega : S \to \Gamma$), l'automate correspond au "modèle de Moore". Sinon, si ω est une fonction de l'état et de l'alphabet d'entrée ($\omega : S \times \Sigma \to \Gamma$) l'automate correspond au "modèle de Mealy". Les automates de Mealy sont plus pratiques dans la plupart des cas, car leur conception permet d'obtenir des automates avec un nombre minimal d'états. En pratique, un mélange des deux modèles est souvent employé.

Malentendu sur les automates finis Il y a un phénomène appelé "explosion d'états", qui est souvent utilisé comme argument défavorable contre l'usage général des automates finis dans les environnements complexes. Il faut mentionner que ce point est trompeur [10]. Les explosions d'états sont souvent le résultat d'une mauvaise conception de l'automate : les erreurs courantes sont de stocker la valeur présente de toutes les entrées dans un état, ou de ne pas diviser un automate complexe en sous-automates plus simples. Le maître EtherCAT utilise plusieurs automates, qui sont exécutés de manière hiérarchique et qui servent de sous-automates. Ils sont aussi décrits ci-dessous.

5.2 Le modèle d'état du maître

Cette section présente les techniques utilisées dans le maître pour implémenter les automates.

Programmation des automates Il y a plusieurs manière d'implémenter un automate avec du code *C*. La manière évidente est d'implémenter les différents états et actions avec un branchement à choix multiple (switch) :

```
1 enum {STATE_1, STATE_2, STATE_3};
2 int state = STATE_1;
3
4 void state_machine_run(void *priv_data) {
5         switch (state) {
6                 case STATE_1:
7                         action_1();
8                         state = STATE_2;
9                         break;
```

```
10                  case STATE_2:
11                          action_2()
12                          if (some_condition) state = STATE_1;
13                          else state = STATE_3;
14                          break;
15                  case STATE_3:
16                          action_3();
17                          state = STATE_1;
18                          break;
19          }
20  }
```

Cette technique reste possible pour les petits automates, mais présente l'inconvénient de complexifier rapidement le code lorsque le nombre d'états augmente. De plus le branchement à choix multiple doit être exécuté à chaque itération et beaucoup d'indentations sont gaspillés.

La méthode retenue par le maître est d'implémenter chaque état dans sa propre fonction et de stocker la fonction d'état courante dans un pointeur de fonction :

```
1  void (*state)(void *) = state1;
2
3  void state_machine_run(void *priv_data) {
4          state(priv_data);
5  }
6
7  void state1(void *priv_data) {
8          action_1();
9          state = state2;
10 }
11
12 void state2(void *priv_data) {
13         action_2();
14         if (some_condition) state = state1;
15         else state = state2;
16 }
17
18 void state3(void *priv_data) {
19         action_3();
20         state = state1;
21 }
```

Dans le code du maître, les pointeurs d'état de tous les automates[3] sont rassemblés dans un objet unique de la classe `ec_fsm_master_t`. C'est avantageux, car il y a

3. Tous sauf l'automate EoE, parce plusieurs esclaves Eoe doivent être gérés en parallèle. Pour cette raison, chaque objet gestionnaire EoE a son propre pointeur d'état.

toujours une instance disponible de chaque automate qui peut être démarrée à la demande.

Mealy et Moore Une vue rapprochée du code ci-dessus montre que les actions exécutées (les "sorties" de l'automate) dépendent uniquement de l'état courant. Ceci correspond au modèle de "Moore" introduit dans section 5.1. Comme déjà mentionné, le modèle de "Mealy" offre une flexibilité supérieure, visible dans le code ci-dessous :

```
1  void state7(void *priv_data) {
2          if (some_condition) {
3                  action_7a();
4                  state = state1;
5          }
6          else {
7                  action_7b();
8                  state = state8;
9          }
10 }
```

③ + ⑦ la fonction d'état exécute les actions en fonction de la transition d'état, qui est sur le point d'être effectuée.

L'alternative la plus flexible est d'exécuter certaines actions en fonction de l'état, puis d'autres actions en fonction de la transition d'état :

```
1  void state9(void *priv_data) {
2          action_9();
3          if (some_condition) {
4                  action_9a();
5                  state = state7;
6          }
7          else {
8                  action_9b();
9                  state = state10;
10         }
11 }
```

Ce modèle est souvent utilisé dans le maître. Il combine les meilleurs aspects des deux approches.

Utilisation de sous-automates Pour éviter d'avoir trop d'états, certaines fonctions de l'automate du maître EtherCAT ont été extraites vers des sous-automates. Ceci améliore l'encapsulation des flux de travail concernés et surtout évite le phénomène d'"explosion d'états" décrit dans section 5.1. Si le maître utilisait à la place un seul gros automate, le nombre d'état serait démultiplié. Ce qui augmenterait le niveau de complexité jusqu'à un niveau ingérable.

Exécution de sous-automates Si un automate démarre l'exécution d'un sous-automate, il reste habituellement dans un état jusqu'à ce que le sous-automate termine son exécution. Ceci est généralement fait comme dans l'extrait de code ci-dessous, qui provient du code de l'automate de configuration des esclaves :

```
1  void ec_fsm_slaveconf_safeop(ec_fsm_t *fsm)
2  {
3          fsm->change_state(fsm); // execute state change
4                                  // sub state machine
5
6          if (fsm->change_state == ec_fsm_error) {
7                  fsm->slave_state = ec_fsm_end;
8                  return;
9          }
10
11         if (fsm->change_state != ec_fsm_end) return;
12
13         // continue state processing
14         ...
```

(3) `change_state` est le pointeur d'état de l'automate. La fonction d'état, sur laquelle pointe le pointeur, est exécutée ...

(6) ... jusqu'à ce que l'automate termine par l'état d'erreur ...

(11) ... ou jusqu'à ce que l'automate termine dans l'état de fin. Pendant ce temps, l'automate "supérieur" reste dans l'état courant et exécute à nouveau le sous-automate dans le prochain cycle.

Description des automates Les sections ci-dessous décrivent chaque automate utilisé par le maître EtherCAT. Les descriptions textuelles des automates contiennent des références aux transitions dans les diagrammes de transitions d'états correspondants, qui sont marqués avec une flèche suivie par le nom de l'état successeur. Les transitions provoquées par des cas d'erreurs triviales (c'est-à-dire, pas de réponse de l'esclave) ne sont pas décrites explicitement. Ces transitions sont décrites sous forme de flèches en tirets dans les diagrammes.

5.3 L'automate du maître

L'automate du maître s'exécute dans le contexte du fil d'exécution (thread) du maître. La figure 5.2 montre son diagramme de transition. Ses buts sont :

Surveillance du bus La topologie du bus est surveillée. Si elle change, le bus est à nouveau analysé.

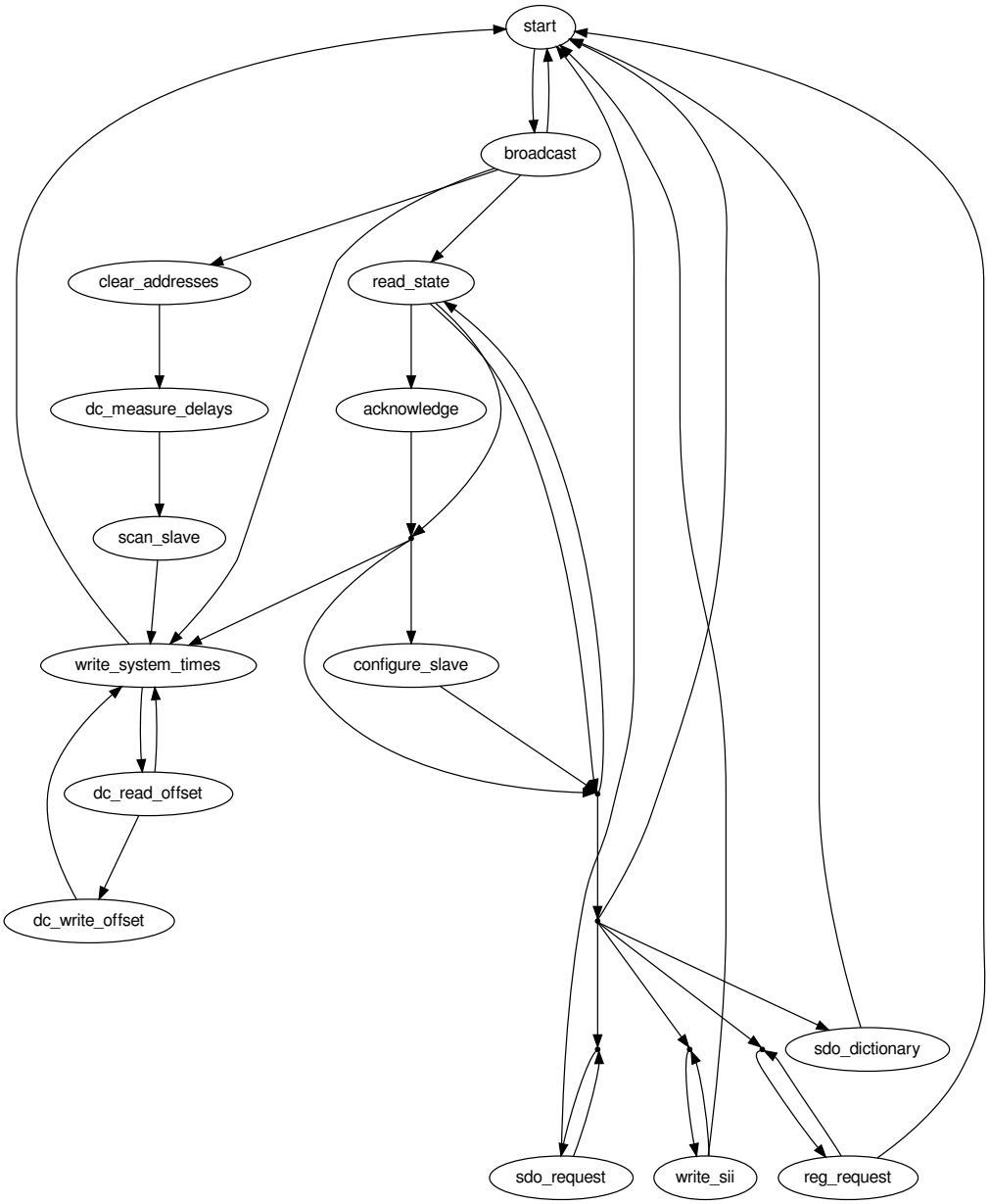

FIGURE 5.2 – Diagramme de transition de l'automate du maître

Configuration des esclaves Les états de la couche application des esclaves sont surveillés. Si un esclave n'est pas dans l'état supposé, alors l'esclave est (re)configuré.

Gestion des requêtes Les requêtes (qui proviennent soit de l'application ou bien de sources externes) sont gérées. Une requête est un travail que le maître traitera de manière asynchrone, par exemple un accès SII, un accès SDO ou similaire.

5.4 L'automate d'analyse des esclaves

L'automate d'analyse des esclaves, qui est représenté dans figure 5.3, conduit le processus de lecture des informations des esclaves.

Le processus d'analyse comprend les étapes suivantes :

Node Address L'adresse du nœud est définie pour l'esclave, de sorte qu'il puisse être adressé par nœud pour toutes les opérations suivantes.

AL State L'état initial de la couche application (Application Layer) est lu.

Base Information L'information de base (tel que le nombre de FMMUs supportées) est lue depuis la mémoire physique la plus basse.

Data Link L'information sur les ports physiques est lue.

SII Size La taille des contenus SII est déterminée pour allouer l'image mémoire SII.

SII Data Les contenus SII sont lus dans l'image du maître.

PREOP Si l'esclave supporte CoE, son état est défini à PREOP en utilisant l'automate de changement d'état (voir section 5.6) pour autoriser la communication par boîte aux lettres et lire la configuration PDO via CoE.

PDOs Les PDOs sont lus via CoE (si supporté) en utilisant l'automate de lecture des PDO (voir section 5.8). Si cela réussit, les informations PDO du SII sont (le cas échéant) écrasées.

5.5 L'automate de configuration de l'état de l'esclave

L'automate de configuration de l'état de l'esclave, qui est représenté dans figure 5.4, configure un esclave et l'amène dans un état particulier de la couche application.

INIT L'automate de changement d'état est utilisé pour amener l'esclave à l'état INIT.

FMMU Clearing Pour éviter que l'esclave réagisse à n'importe quelle donnée de processus, la configuration FMMU est effacée. Si l'esclave ne supporte pas les FMMUs, cet état est sauté. Si INIT est l'état demandé, l'automate est terminé.

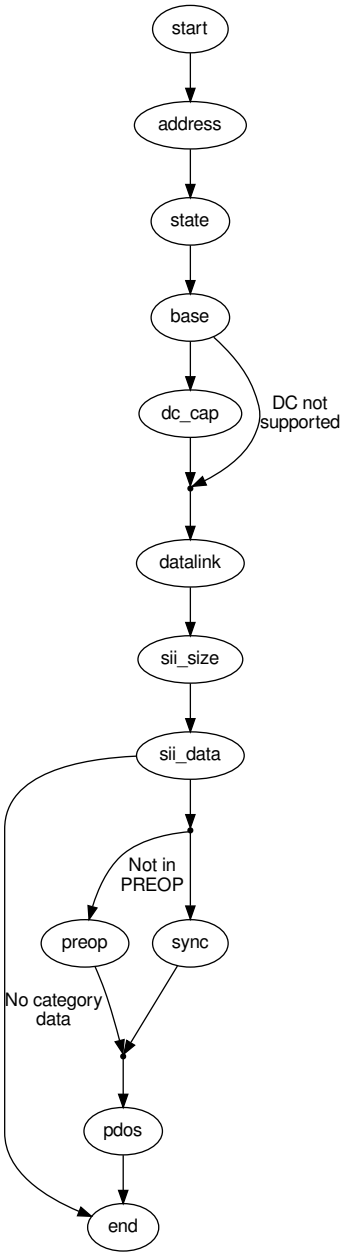

FIGURE 5.3 – Diagramme de transition de l'automate d'analyse des esclaves

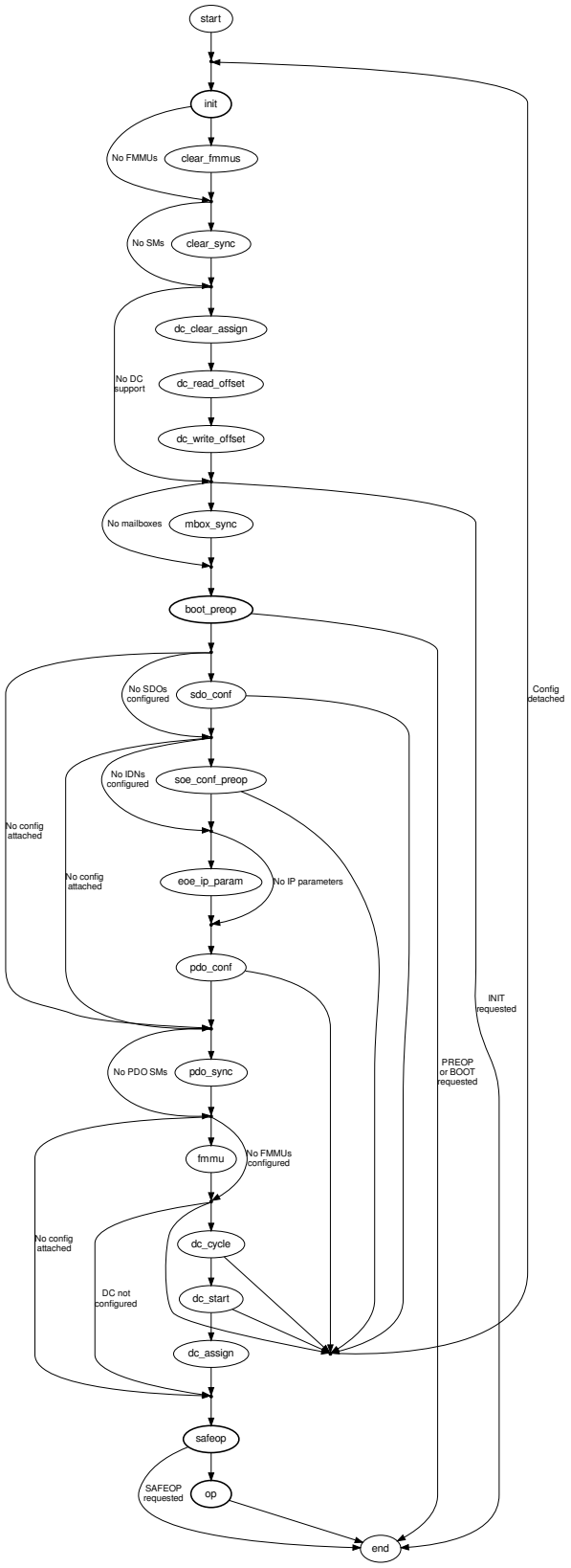

FIGURE 5.4 – Diagramme de transition de l'automate de configuration de l'état de l'esclave

Mailbox Sync Manager Configuration Si l'esclave supporte la communication par boîte aux lettres, les gestionnaires de synchronisation des boîtes aux lettres sont configurés. Sinon cet état est sauté.

PREOP L'automate de changement d'état est utilisé pour amener l'esclave à l'état PREOP. Si PREOP est l'état demandé, l'automate est terminé.

SDO Configuration Si une configuration d'esclave est attachée (voir section 3.1), et que l'application fournit des configurations SDO, elles sont envoyées à l'esclave.

PDO Configuration L'automate de configuration PDO est exécuté pour appliquer toutes les configurations PDO nécessaires.

PDO Sync Manager Configuration S'il y a des gestionnaires de synchronisation PDO, ils sont configurés.

FMMU Configuration Si l'application fournit des configurations FMMU (i. e. si l'application a inscrit des entrées PDO), elles sont appliquées.

SAFEOP L'automate de changement d'état est utilisé pour amener l'esclave à l'état SAFEOP. Si SAFEOP est l'état demandé, l'automate est terminé.

OP L'automate de changement d'état est utilisé pour amener l'esclave à l'état OP. Si OP est l'état demandé, l'automate est terminé.

5.6 L'automate de changement d'état

L'automate de changement d'état, qui est représenté dans figure 5.5, conduit le processus de changement d'état de la couche application de l'esclave. Il implémente les états et transitions décrits dans [3, sec. 6.4.1].

Start Le nouvel état de la couche d'application (AL : application-layer) est demandé via le registre "AL Control Request" (voir [3, sec. 5.3.1]).

Check for Response Certains esclaves ont besoin de temps pour répondre à une commande de changement d'état AL et ne répondent pas pendant un certain temps. Dans ce cas, la commande est à nouveau émise, jusqu'à l'accusé de réception.

Check AL Status Si le datagramme de changement d'état AL a été acquité, le registre "AL Control Response" (voir [3, sec. 5.3.2]) doit être lu jusqu'à ce que l'esclave change l'état AL.

AL Status Code Si l'esclave refuse la commande de changement d'état, la raison peut être lue dans le champ "AL Status Code" des registres "AL State Changed" (voir [3, sec. 5.3.3]).

Acknowledge State Si le changement d'état n'a pas réussi, le maître doit accuser réception de l'ancien état en écrivant à nouveau dans le registre "AL Control request".

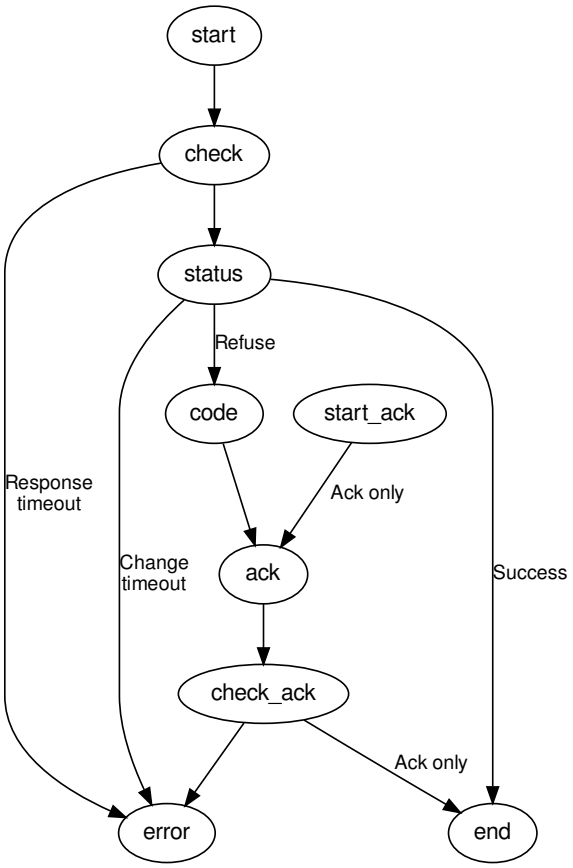

FIGURE 5.5 – Diagramme de transition de l'automate de changement d'état

Check Acknowledge Après l'envoi de la commande d'accusé de réception, le registre "AL Control Response" doit être lu à nouveau.

L'état "start_ack" est un raccourci dans l'automate quand le maître veut accuser réception d'un changement spontané d'état AL, qui n'avait pas été demandé.

5.7 L'automate SII

L'automate SII (présenté dans figure 5.6) implémente le processus de lecture ou d'écriture des données SII via l'interface d'information de l'esclave (Slave Information Interface) décrite dans [2, sec. 6.4].

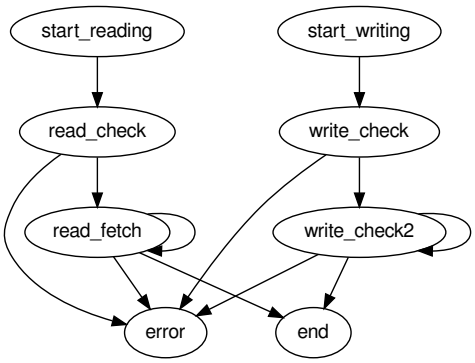

FIGURE 5.6 – Diagramme de transition de l'automate SII

Voici comment fonctionne la partie lecture de l'automate :

Start Reading La requête de lecture et l'adresse du mot demandé sont écrits dans l'attribut SII.

Check Read Command Si la commande de lecture SII a reçu son accusé de réception, un chronomètre est démarré. Un datagramme est envoyé pour lire l'attribut SII pour l'état et les données.

Fetch Data Si l'opération de lecture est encore en attente (la SII est habituellement implémentée avec une E^2PROM), l'état est lu à nouveau. Sinon les données sont copiées dans le datagramme.

La partie écriture est presque similaire :

Start Writing Une requête d'écriture, l'adresse destination et le mot de donnée sont écrits dans l'attribut SII.

Check Write Command Si la commande d'écriture SII a reçu son accusé de réception, un chronomètre est démarré. Un datagramme est envoyé pour lire l'attribut SII pour l'état de l'opération d'écriture.

Wait while Busy Si l'opération d'écriture est encore en attente (déterminé par un temps d'attente minimal et l'état du drapeau busy), l'automate reste dans cet état pour éviter qu'une autre opération d'écriture ne soit émise trop tôt.

5.8 Les automates PDO

Les automates PDO sont un ensemble d'automates qui lisent ou écrivent l'affectation PDO et la cartographie des PDO via la "zone de communication CoE" décrite dans [3, sec. 5.6.7.4]. Pour l'accès aux objets, les primitives CANopen over EtherCAT sont utilisées (voir section 6.2), donc l'esclave doit obligatoirement supporter le protocole de boîte aux lettres CoE.

Automate de lecture PDO Cet automate (figure 5.7) a pour but de lire la configuration PDO complète d'un esclave. Il lit l'affectation PDO et pour chaque gestionnaire de configuration il utilise l'automate de lecture des entrées PDO (figure 5.8) pour lire la cartographie de chaque PDO assigné.

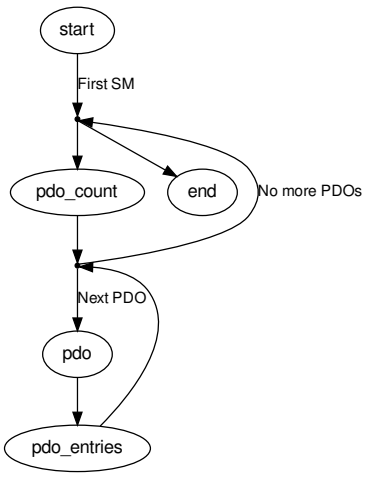

FIGURE 5.7 – Diagramme de transition de l'automate de lecture des PDO

Fondamentalement, il lit pour chaque gestionnaire de synchronisation, le compteur de PDOs affectés à ce gestionnaire de synchronisation via l'objet SDO `0x1C1x`. Il lit ensuite les sous-index du SDO pour obtenir les indices des PDO affectés. Quand un index PDO est lu, l'automate de lecture des entrées PDO est exécuté pour lire les entrées PDO qui sont mappées en mémoire.

L'automate de lecture des entrées PDO Cet automate (figure 5.8) lit la cartograhie PDO (les entrées PDO) d'un PDO. Il lit la cartographie SDO respective (`0x1600` – `0x17ff`, ou `0x1a00` – `0x1bff`) pour le PDO donné en lisant le sous-index zéro (nombre d'éléments) pour déterminer le nombre d'entrée PDO projetés en mémoire. Après cela, chaque sous-index est lu pour obtenir l'index de l'entrée PDO mappée en mémoire, ainsi que son sous-index et sa taille en bits.

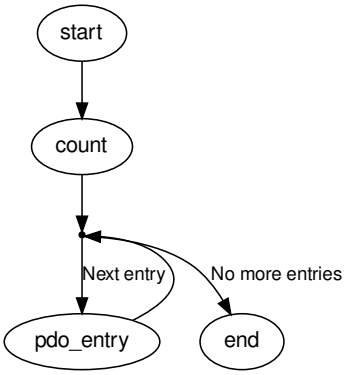

FIGURE 5.8 – Diagramme de transition de l'automate de lecture des entrées PDO

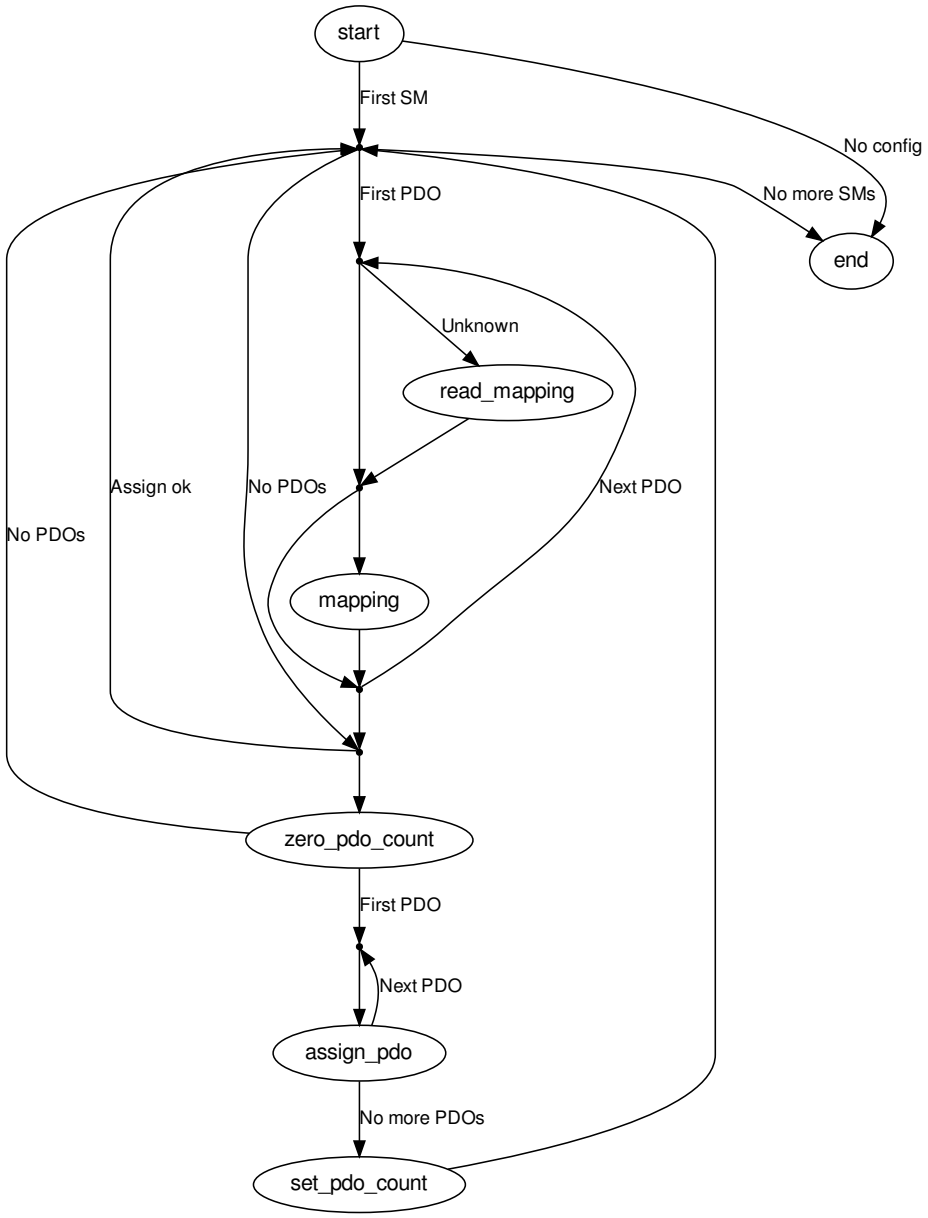

FIGURE 5.9 – Diagramme de transition de l'automate de configuration des PDO

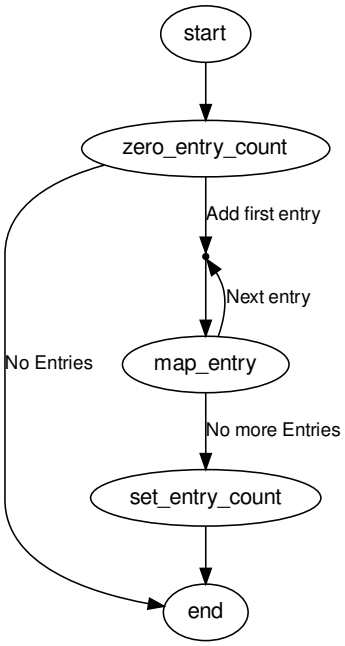

FIGURE 5.10 – Diagramme de transition de l'automate de configuration des entrées PDO

6 Implémentation du protocole de boîte aux lettres

Le maître EtherCAT implémente les protocoles de boîte aux lettres CANopen over EtherCAT (CoE), Ethernet over EtherCAT (EoE), File-access over EtherCAT (FoE), Vendor-specific over EtherCAT (VoE) et Servo Profile over EtherCAT (SoE). Voir les sections ci-dessous pour les détails.

6.1 Ethernet over EtherCAT (EoE)

Le maître EtherCAT implémente le protocole de boîte aux lettres Ethernet over EtherCAT [3, sec. 5.7] pour permettre le tunnelage de trames Ethernet vers des esclaves spéciaux, qui peuvent soit avoir des ports physiques Ethernet ou avoir leur propre pile IP pour recevoir les trames.

Interfaces réseaux virtuelles Le maître crée une interface réseau virtuelle EoE pour chaque esclave compatible EoE. Ces interface sont nommées

eoeXsY pour un esclave sans adresse alias (voir sous-section 7.1.2), où X est l'index du maître et Y la position de l'esclave sur l'anneau.

eoeXaY pour un esclave avec une adresse d'alias non-nulle, où X est l'index du maître et Y est l'adresse alias en décimal.

Les trames envoyées vers ces interfaces sont transférées vers les esclaves associés par le maître. Les trames reçues par les esclaves sont récupérées par le maître et transférées aux interfaces virtuelles.

Ceci apporte les avantages suivants :

— Flexibilité : l'utilisateur peut décider comment les esclaves compatibles EoE sont interconnectés avec le reste du monde.
— Les outils standards peuvent être utilisés pour surveiller l'activité EoE et pour configurer les interfaces EoE.
— L'implémentation du pontage de niveau 2 du noyau Linux (selon la norme de pontage IEEE 802.1D MAC) peut être utilisée nativement pour relier le trafic Ethernet entre les esclaves compatibles EoE.
— La pile réseau du noyau Linux peut être utilisée pour router les paquets entre les esclaves compatibles EoE et pour suivre les problèmes de sécurité, comme avec une interface réseau physique.

EoE Handlers Les interface virtuelles EoE et les fonctionnalités relatives sont encapsulées dans la classe `ec_eoe_t` class. Un objet de cette classe est appelé "gestionnaire EoE". Par exemple, le maître ne crée pas les interfaces réseaux directement : ceci est fait à l'intérieur du constructeur d'un gestionnaire EoE. Un gestionnaire EoE contient également une file d'attente pour les trames. À chaque fois que le noyau passe un nouveau tampon de socket pour l'envoyer via la fonction de rappel `hard_start_xmit()` de l'interface, le tampon de socket est mis en file d'attente pour la transmission via l'automate EoE (voir ci-dessous). Si la file d'attente est pleine, le passage des nouveaux tampons de socket est suspendu par un appel à `netif_stop_queue()`.

Création de gestionnaire EoE Pendant l'analyse du bus (voir section 5.4), le maître détermine les protocoles de boîte aux lettres supportés par chaque esclave. Ceci est fait en examinant le champ de bits "Protocoles de boîte aux lettres supportés" au mot d'adresse 0x001C de la SII. Si le bit 1 est défini, alors l'esclave supporte le protocole EoE. Dans ce cas, un gestionnaire EoE est créé pour cet esclave.

Automate EoE Chaque gestionnaire EoE possède son automate EoE, qui est utilisé pour envoyer des trames à l'esclave correspondant et recevoir des trames de celui-ci via les primitives de communication EoE. Cette automate est présenté dans figure 6.1.

RX_START L'état de départ de l'automate EoE. Un datagramme de vérification de la boîte aux lettres est envoyé pour demander de nouvelles trames à la boîte aux lettres de l'esclave. → RX_CHECK

RX_CHECK Le datagramme de vérification de la boîte aux lettres est reçu. Si la boîte aux lettres de l'esclave ne contenait pas de données, un cycle de transmission débute. → TX_START

S'il y a des nouvelles données dans la boîte aux lettres, un datagramme est envoyé pour rapatrier les nouvelles données. → RX_FETCH

RX_FETCH Le datagramme de rapatriement est reçu. Si la donnée dans la boîte aux lettres ne contient pas de commande de "requête de fragment EoE", les données sont abandonnées et une séquence de transmission démarre. → TX_START

Si la trame Ethernet reçue est le premier fragment, un nouveau tampon de socket est alloué. Sinon, les données sont copiées à la bonne position dans le tampon de socket.

Si le fragment est le dernier fragment, le tampon de socket est envoyé à la pile réseau et une séquence de transmission est démarrée. → TX_START

Sinon, une nouvelle séquence de réception est démarrée pour rappatrier le prochain fragment. → RX_START

TX_START L'état de démarrage de la séquence de transmission. On vérifie si la file d'attente de la transmission contient une trame à envoyer. Sinon, une séquence de réception est démarrée → RX_START

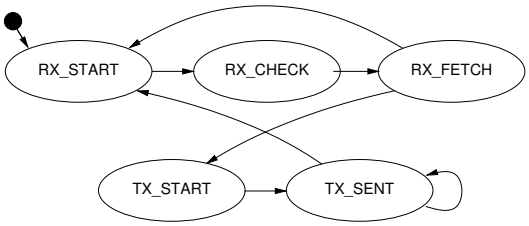

FIGURE 6.1 – Diagramme de transition de l'automate EoE

S'il y a une trame à envoyer, elle est retirée de la file d'attente. Si la file d'attente était inactive auparavant (parce qu'elle était pleine), la file d'attente est réveillée par un appel à *netif_wake_queue()*. Le premier fragment de la trame est envoyé. → TX_SENT

TX_SENT On vérifie si le premier fragment a été envoyé avec succès. Si la trame actuelle est constituée de fragments supplémentaires, le prochain est envoyé. → TX_SENT

Si le dernier fragment a été envoyé, une nouvelle séquence de réception est démarrée. → RX_START

Traitement EoE Pour exécuter l'automate EoE de chaque gestionnaire EoE actif, il doit y avoir un processus cyclique. La solution la plus simple serait d'exécuter les automates EoE de manière synchrone avec l'automate du maître (voir section 5.3. Cette approche a les inconvénients suivants :

Un seul fragment EoE pourrait être envoyé ou reçu tous les quelques cycles. Le débit des données serait très faible, parce que les automates EoE ne seraient pas exécutés entre les cycles de l'application. En outre, le débit dépendrait de la période de la tâche application.

Pour surmonter ce problème, les automates EoE ont besoin de leur propre processus cyclique pour s'exécuter. Pour cela, le maître possède un timer noyau, qui est exécuté à chaque interruption temporelle. Ceci garantie une bande passante constante, mais pose un nouveau problème d'accès concurrent au maître. Le mécanisme de verrouillage nécessaire à cet effet est présenté dans section 3.4.

Configuration automatique Par défaut, les esclaves sont laissés dans l'état PREOP si aucune configuration n'est appliquée. Si le lien de l'interface EoE est configuré à "up", l'état de la couche application de l'esclave concerné passe automatiquement à OP.

6.2 CANopen over EtherCAT (CoE)

Le protocole CANopen over EtherCAT [3, sec. 5.6] permet de configurer les esclaves et d'échanger des objets de données au niveau de l'application.

Automate de téléchargement SDO Le meilleur moment pour appliquer les configurations SDO est pendant l'état PREOP, parce que la communication par boîte aux lettres est déjà possible et que l'application de l'esclave va démarrer avec la mise à jour des données d'entrées dans le prochain état SAFEOP. C'est pourquoi, la configuration SDO doit faire partie de l'automate de configuration de l'esclave (voir section 5.5) : ceci est implémenté via l'automate de téléchargement SDO, qui est exécuté juste

avant que l'esclave entre dans l'état SAFEOP. De cette manière, il est garanti que les configurations SDO soient appliquées à chaque fois que l'esclave est reconfiguré.

Le diagramme de transition de l'automate de téléchargement SDO est présenté dans figure 6.2.

START L'état de départ de l'automate de téléchargement CoE. La commande de boîte aux lettres "SDO Download Normal Request" est envoyée. → REQUEST

REQUEST On vérifie que l'esclave a reçu la requête de téléchargement CoE. Après cela, la commande de vérification de la boîte aux lettres est émise et un minuteur est lancé. → CHECK

CHECK Si aucune donnée n'est disponible dans la boîte aux lettres, le minuteur est vérifié.

— S'il a expiré, le téléchargement SDO est interrompu. → ERROR
— Sinon la boîte aux lettres est à nouveau interrogée. → CHECK

Si la boîte aux lettres contient des nouvelles données, la réponse est rapatriée. → RESPONSE

RESPONSE Si la réponse de la boîte aux lettres ne peut pas être récupérée, c'est que les données sont invalides, ou qu'on a reçu le mauvais protocole ou un "Abort SDO Transfer Request". Alors on arrête le téléchargement SDO. → ERROR

Si on reçoit l'accusé de réception "SDO Download Normal Response", le téléchargement SDO a réussi. → END

END Le téléchargement SDO a réussi.

ERROR Une erreur a arrêté le téléchargement SDO.

6.3 Vendor specific over EtherCAT (VoE)

Le protocole VoE permet d'implémenter des protocoles de communication par boîte aux lettres spécifiques pour un fabricant. Les messages VoE sont préfixés par un entête VoE qui contient l'identité du fabricant (vendor ID) sur 32 bits et le type de fabricant (vendor-type) sur 16 bit. Il n'y a aucune autre contrainte pour ce protocole.

Le maître EtherCAT autorise la création multiple de gestionnaires VoE pour les configurations d'esclaves via l'API (voir chapitre 3). Ces gestionnaires contiennent les automates nécessaires à la communication via VoE.These

Pour davantage d'information sur les gestionnaires VoE, voir section 3.3 ou les applications d'exemples dans le sous-dossier *examples/*.

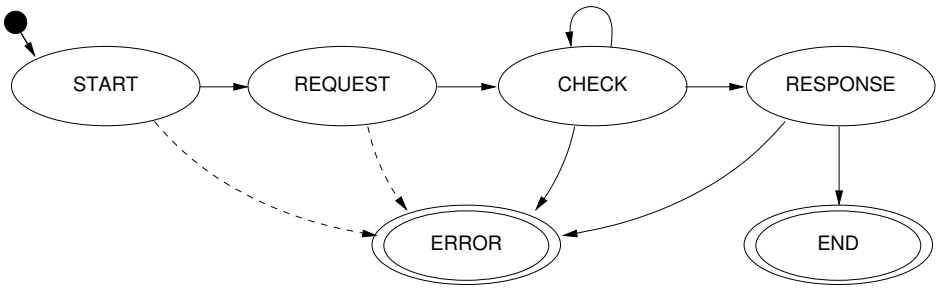

FIGURE 6.2 – Diagramme de transition de l'automate de téléchargement CoE

6.4 Servo Profile over EtherCAT (SoE)

Le protocole SoE implémente la couche canal de service, spécifiée dans IEC 61800-7 [16] via les boîtes aux lettres EtherCAT.

Le protocole SoE est très similaire au protocole CoE (vor section 6.2). Mais à la place des index et sous-index SDO, des numéros d'identification (IDNs) identifient les paramètres.

L'implémentation couvre les primitives "SCC Read" et "SCC Write", chacune est capable de fragmenter les données.

Il y a plusieurs manières d'utiliser l'implémentation SoE :

— Lecture et écriture des IDNs via l'outil en ligne de commande (voir sous-section 7.1.18).
— Stocker des configuration pour des IDNs arbitraires via l'API (voir chapitre 3, i. e. `ecrt_slave_config_idn()`). Ces configurations sont écrites dans l'esclave pendant la configuration dans l'état PREOP, avant de passer en SAFEOP.
— La bibliothèque en espace utilisateur (voir section 7.2), offre des fonctions pour lire/écrire les IDNs en mode bloquant (`ecrt_master_read_idn()`, `ecrt_master_write_idn()`).

7 Interfaces dans l'espace utilisateur

Puisque le maître s'exécute en tant que module noyau, ses accès natifs se limitent à analyser les messages Syslog et à le contrôler avec *modutils*.

Il était donc nécessaire d'implémenter d'autres interface pour faciliter l'accès au maître depuis l'espace utilisateur et pour permettre une influence plus fine. Il doit être possible de voir et de changer des paramètres spéciaux en cours d'exécution.

La visualisation du bus est un autre point : dans un but de développement et de déverminage, il est nécessaire, par exemple, de montrer les esclaves connectés (voir section 7.1).

L'API doit être disponible depuis l'espace utilisateur pour permettre aux programmes qui s'y trouvent d'utiliser les fonctionnalités EtherCAT. Ceci est implémenté via un périphérique en mode caractère et une bibliothèque en espace utilisateur (voir section 7.2).

Le démarrage et la configuration automatique sont d'autres aspects. Le maître doit être capable de démarrer automatiquement avec une configuration persistante (voir section 7.4).

La surveillance des communications EtherCAT est un dernier point. Dans un but de déverminage, il faut avoir un moyen d'analyser les datagrammes EtherCAT. La meilleure solution serait d'utiliser un analyseur réseau populaire, tel que Wireshark [8] ou d'autres (voir section 7.5).

Ce chapitre couvre tous ces points et présente les interfaces et outils qui les rendent possibles.

7.1 Outil en ligne de commande

7.1.1 Périphériques en mode caractères

Chaque instance de maître recoit un périphérique en mode caractère comme interface en espace utilisateur. Les périphériques sont nommés */dev/EtherCATx*, où $x \in \{0 \dots n\}$ est l'index du maître.

Création des nœuds de périphériques Les nœuds des périphériques en mode caractères sont automatiquement créés si le paquet `udev` est installé. Voir section 9.5 pour son installation et sa configuration.

7.1.2 Paramètre d'alias d'adresse

```
ethercat alias [OPTIONS] <ALIAS>

Write alias addresses.

Arguments:
  ALIAS must be an unsigned 16 bit number. Zero means
        removing an alias address.

If multiple slaves are selected, the --force option
is required.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --force    -f          Acknowledge writing aliases of
                         multiple slaves.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.3 Affichage de la configuration du bus

```
ethercat config [OPTIONS]

Show slave configurations.

Without the --verbose option, slave configurations are
output one-per-line. Example:

1001:0  0x0000003b/0x02010000  3  OP
|       |                      |  |
|       |                      |  \- Application-layer
|       |                      |     state of the attached
|       |                      |     slave, or '-', if no
|       |                      |     slave is attached.
|       |                      \- Absolute decimal ring
|       |                         position of the attached
|       |                         slave, or '-' if none
|       |                         attached.
|       \- Expected vendor ID and product code (both
|          hexadecimal).
\- Alias address and relative position (both decimal).

With the --verbose option given, the configured PDOs and
SDOs are output in addition.

Configuration selection:
  Slave configurations can be selected with
```

```
  the --alias and --position parameters as follows:

  1) If neither the --alias nor the --position option
     is given, all slave configurations are displayed.
  2) If only the --position option is given, an alias
     of zero is assumed (see 4)).
  3) If only the --alias option is given, all slave
     configurations with the given alias address
     are displayed.
  4) If both the --alias and the --position option are
     given, the selection can match a single
     configuration, that is displayed, if it exists.

Command-specific options:
  --alias    -a <alias>  Configuration alias (see above).
  --position -p <pos>    Relative position (see above).
  --verbose  -v          Show detailed configurations.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.4 Sortie des informations PDO en langage C

```
ethercat cstruct [OPTIONS]

Generate slave PDO information in C language.

The output C code can be used directly with the
ecrt_slave_config_pdos() function of the application
interface.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.5 Affichage des données de processus

```
ethercat data [OPTIONS]

Output binary domain process data.

Data of multiple domains are concatenated.

Command-specific options:
  --domain -d <index>  Positive numerical domain index.
                       If omitted, data of all domains
                       are output.
```

```
Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.6 Configuration du niveau de déverminage d'un maître

```
ethercat debug <LEVEL>

Set the master's debug level.

Debug messages are printed to syslog.

Arguments:
  LEVEL can have one of the following values:
        0 for no debugging output,
        1 for some debug messages, or
        2 for printing all frame contents (use with caution!).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.7 Domaines configurés

```
ethercat domains [OPTIONS]

Show configured domains.

Without the --verbose option, the domains are displayed
one-per-line. Example:

Domain0: LogBaseAddr 0x00000000, Size   6, WorkingCounter 0/1

The domain's base address for the logical datagram
(LRD/LWR/LRW) is displayed followed by the domain's
process data size in byte. The last values are the current
datagram working counter sum and the expected working
counter sum. If the values are equal, all PDOs were
exchanged during the last cycle.

If the --verbose option is given, the participating slave
configurations/FMMUs and the current process data are
additionally displayed:

Domain1: LogBaseAddr 0x00000006, Size   6, WorkingCounter 0/1
  SlaveConfig 1001:0, SM3 ( Input), LogAddr 0x00000006, Size 6
    00 00 00 00 00 00

The process data are displayed as hexadecimal bytes.

Command-specific options:
  --domain  -d <index>  Positive numerical domain index.
```

```
                        If omitted, all domains are
                        displayed.

  --verbose -v          Show FMMUs and process data
                        in addition.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.8 Accès SDO

```
ethercat download [OPTIONS] <INDEX> <SUBINDEX> <VALUE>
 [OPTIONS] <INDEX> <VALUE>

Write an SDO entry to a slave.

This command requires a single slave to be selected.

The data type of the SDO entry is taken from the SDO
dictionary by default. It can be overridden with the
--type option. If the slave does not support the SDO
information service or the SDO is not in the dictionary,
the --type option is mandatory.

The second call (without <SUBINDEX>) uses the complete
access method.

These are valid data types to use with
the --type option:
  bool,
  int8, int16, int32, int64,
  uint8, uint16, uint32, uint64,
  float, double,
  string, octet_string, unicode_string.
For sign-and-magnitude coding, use the following types:
  sm8, sm16, sm32, sm64

Arguments:
  INDEX    is the SDO index and must be an unsigned
           16 bit number.
  SUBINDEX is the SDO entry subindex and must be an
           unsigned 8 bit number.
  VALUE    is the value to download and must correspond
           to the SDO entry datatype (see above). Use
           '-' to read from standard input.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --type     -t <type>   SDO entry data type (see above).
```

```
Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.

ethercat upload [OPTIONS] <INDEX> <SUBINDEX>

Read an SDO entry from a slave.

This command requires a single slave to be selected.

The data type of the SDO entry is taken from the SDO
dictionary by default. It can be overridden with the
--type option. If the slave does not support the SDO
information service or the SDO is not in the dictionary,
the --type option is mandatory.

These are valid data types to use with
the --type option:
  bool,
  int8, int16, int32, int64,
  uint8, uint16, uint32, uint64,
  float, double,
  string, octet_string, unicode_string.
For sign-and-magnitude coding, use the following types:
  sm8, sm16, sm32, sm64

Arguments:
  INDEX    is the SDO index and must be an unsigned
           16 bit number.
  SUBINDEX is the SDO entry subindex and must be an
           unsigned 8 bit number.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --type     -t <type>   SDO entry data type (see above).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.9 Statistiques EoE

```
ethercat eoe

Display Ethernet over EtherCAT statictics.

The TxRate and RxRate are displayed in Byte/s.
```

7.1.10 File-Access over EtherCAT

```
ethercat foe_read [OPTIONS] <SOURCEFILE>
```

```
Read a file from a slave via FoE.

This command requires a single slave to be selected.

Arguments:
  SOURCEFILE is the name of the source file on the slave.

Command-specific options:
  --output-file -o <file>   Local target filename. If
                            '-' (default), data are
                            printed to stdout.
  --alias       -a <alias>
  --position    -p <pos>    Slave selection. See the help
                            of the 'slaves' command.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.


ethercat foe_write [OPTIONS] <FILENAME>

Store a file on a slave via FoE.

This command requires a single slave to be selected.

Arguments:
  FILENAME can either be a path to a file, or '-'. In
           the latter case, data are read from stdin and
           the --output-file option has to be specified.

Command-specific options:
  --output-file -o <file>   Target filename on the slave.
                            If the FILENAME argument is
                            '-', this is mandatory.
                            Otherwise, the basename() of
                            FILENAME is used by default.
  --alias       -a <alias>
  --position    -p <pos>    Slave selection. See the help
                            of the 'slaves' command.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.11 Création de graphiques topologiques

```
ethercat graph [OPTIONS]
ethercat graph [OPTIONS] <INFO>

Output the bus topology as a graph.

The bus is output in DOT language (see
```

```
http://www.graphviz.org/doc/info/lang.html), which can
be processed with the tools from the Graphviz
package. Example:

  ethercat graph | dot -Tsvg > bus.svg

See 'man dot' for more information.

Additional information at edges and nodes is selected via
the first argument:
  DC  - DC timing
  CRC - CRC error register information
```

7.1.12 Maître et périphériques Ethernet

```
ethercat master [OPTIONS]

Show master and Ethernet device information.

Command-specific options:
  --master -m <indices>  Master indices. A comma-separated
                         list with ranges is supported.
                         Example: 1,4,5,7-9. Default: - (all).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.13 Gestionnaires de synchronisation, PDOs et entrées PDO

```
ethercat pdos [OPTIONS]

List Sync managers, PDO assignment and mapping.

For the default skin (see --skin option) the information
is displayed in three layers, which are
indented accordingly:

1) Sync managers - Contains the sync manager information
   from the SII: Index, physical start address, default
   size, control register and enable word. Example:

   SM3: PhysAddr 0x1100, DefaultSize 0, ControlRegister 0x20, Enable
      1

2) Assigned PDOs - PDO direction, hexadecimal index and
   the PDO name, if available. Note that a 'Tx' and 'Rx'
   are seen from the slave's point of view. Example:

   TxPDO 0x1a00 "Channel1"

3) Mapped PDO entries - PDO entry index and subindex (both
```

```
   hexadecimal), the length in bit and the description, if
   available. Example:

   PDO entry 0x3101:01, 8 bit, "Status"

Note, that the displayed PDO assignment and PDO mapping
information can either originate from the SII or from the
CoE communication area.

The "etherlab" skin outputs a template configuration
for EtherLab's generic EtherCAT slave block.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --skin     -s <skin>   Choose output skin. Possible values are
                         "default" and "etherlab".

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.14 Registre d'accès

```
ethercat reg_read [OPTIONS] <ADDRESS> [SIZE]

Output a slave's register contents.

This command requires a single slave to be selected.

Arguments:
  ADDRESS is the register address. Must
          be an unsigned 16 bit number.
  SIZE    is the number of bytes to read and must also be
          an unsigned 16 bit number. ADDRESS plus SIZE
          may not exceed 64k. The size is ignored (and
          can be omitted), if a selected data type
          implies a size.

These are valid data types to use with
the --type option:
  bool,
  int8, int16, int32, int64,
  uint8, uint16, uint32, uint64,
  float, double,
  string, octet_string, unicode_string.
For sign-and-magnitude coding, use the following types:
  sm8, sm16, sm32, sm64

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
```

```
                          the 'slaves' command.
  --type     -t <type>    Data type (see above).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.


ethercat reg_write [OPTIONS] <OFFSET> <DATA>

Write data to a slave's registers.

This command requires a single slave to be selected.

Arguments:
  ADDRESS is the register address to write to.
  DATA    depends on whether a datatype was specified
          with the --type option: If not, DATA must be
          either a path to a file with data to write,
          or '-', which means, that data are read from
          stdin. If a datatype was specified, VALUE is
          interpreted respective to the given type.

These are valid data types to use with
the --type option:
  bool,
  int8, int16, int32, int64,
  uint8, uint16, uint32, uint64,
  float, double,
  string, octet_string, unicode_string.
For sign-and-magnitude coding, use the following types:
  sm8, sm16, sm32, sm64

Command-specific options:
  --alias     -a <alias>
  --position  -p <pos>    Slave selection. See the help of
                          the 'slaves' command.
  --type      -t <type>   Data type (see above).
  --emergency -e          Send as emergency request.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.15 Dictionnaire SDO

```
ethercat sdos [OPTIONS]

List SDO dictionaries.

SDO dictionary information is displayed in two layers,
which are indented accordingly:

1) SDOs - Hexadecimal SDO index and the name. Example:
```

```
   SDO 0x1018, "Identity object"

2) SDO entries - SDO index and SDO entry subindex (both
   hexadecimal) followed by the access rights (see
   below), the data type, the length in bit, and the
   description. Example:

   0x1018:01, rwrwrw, uint32, 32 bit, "Vendor id"

The access rights are specified for the AL states PREOP,
SAFEOP and OP. An 'r' means, that the entry is readable
in the corresponding state, an 'w' means writable,
respectively. If a right is not granted, a dash '-' is
shown.

If the --quiet option is given, only the SDOs are output.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --quiet    -q          Only output SDOs (without the
                         SDO entries).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.16 Accès SSI

Il est possible de lire ou écrire directement tout le contenu SII des esclaves. Ceci a été ajouté pour les raisons ci-dessous :

- Le format des données SII est encore en développement et des catégories peuvent être ajoutées dans le futur. Avec les accès en lecture et écriture, tout le contenu de la mémoire peut être facilement sauvegardé et restauré.
- Certaines champs SII doivent être altérés (par exemple les alias d'adresses). Une écriture rapide est donc nécessaire pour cela.
- Au travers de l'accès en lecture, l'analyse des catégories de données doit être possible depuis l'espace utilisateur.

```
ethercat sii_read [OPTIONS]

Output a slave's SII contents.

This command requires a single slave to be selected.

Without the --verbose option, binary SII contents are
output.

With the --verbose option given, a textual representation
```

```
of the data is output, that is separated by SII category
names.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --verbose  -v          Output textual data with
                         category names.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

Le lecture des données SII est aussi facile que les autres commandes. Comme les données sont au format binaire, l'analyse est plus facile avec un outil tel que *hexdump* :

```
$ ethercat sii_read --position 3 | hexdump
0000000 0103 0000 0000 0000 0000 0000 0000 008c
0000010 0002 0000 3052 07f0 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
...
```

La sauvegarde de la SII peut être facilement faite avec une redirection :

```
$ ethercat sii_read --position 3 > sii-of-slave3.bin
```

Pour téléverser une SII dans un esclave, l'accès en écriture au périphérique en mode caractère du maître est nécessaire (voir sous-section 7.1.1).

```
ethercat sii_write [OPTIONS] <FILENAME>

Write SII contents to a slave.

This command requires a single slave to be selected.

The file contents are checked for validity and integrity.
These checks can be overridden with the --force option.

Arguments:
  FILENAME must be a path to a file that contains a
           positive number of words. If it is '-',
           data are read from stdin.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --force    -f          Override validity checks.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

```
# ethercat sii_write --position 3 sii-of-slave3.bin
```

La validité du contenu de la SSI peut être vérifiée puis le contenu est envoyé à l'esclave. L'opération d'écriture peut prendre quelques secondes.

7.1.17 Esclaves sur le bus

Les informations sur les esclaves peuvent être collectées avec la sous-commande `slaves` :

```
ethercat slaves [OPTIONS]

Display slaves on the bus.

If the --verbose option is not given, the slaves are
displayed one-per-line. Example:

1  5555:0  PREOP  +  EL3162 2C. Ana. Input 0-10V
|  |     |  |      |  |
|  |     |  |      |  \- Name from the SII if available,
|  |     |  |      |     otherwise vendor ID and product
|  |     |  |      |     code (both hexadecimal).
|  |     |  |      \- Error flag. '+' means no error,
|  |     |  |         'E' means that scan or
|  |     |  |         configuration failed.
|  |     |  \- Current application-layer state.
|  |     \- Decimal relative position to the last
|  |        slave with an alias address set.
|  \- Decimal alias address of this slave (if set),
|     otherwise of the last slave with an alias set,
|     or zero, if no alias was encountered up to this
|     position.
\- Absolute ring position in the bus.

If the --verbose option is given, a detailed (multi-line)
description is output for each slave.

Slave selection:
  Slaves for this and other commands can be selected with
  the --alias and --position parameters as follows:

  1) If neither the --alias nor the --position option
     is given, all slaves are selected.
  2) If only the --position option is given, it is
     interpreted as an absolute ring position and
     a slave with this position is matched.
  3) If only the --alias option is given, all slaves
     with the given alias address and subsequent
     slaves before a slave with a different alias
     address match (use -p0 if only the slaves
     with the given alias are desired, see 4)).
```

```
  4) If both the --alias and the --position option are
     given, the latter is interpreted as relative
     position behind any slave with the given alias.

Command-specific options:
  --alias    -a <alias>  Slave alias (see above).
  --position -p <pos>    Slave position (see above).
  --verbose  -v          Show detailed slave information.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

Voici par exemple une sortie typique :

```
$ ethercat slaves
0      0:0  PREOP  +  EK1100 Ethernet Kopplerklemme (2A E-Bus)
1   5555:0  PREOP  +  EL3162 2K. Ana. Eingang 0-10V
2   5555:1  PREOP  +  EL4102 2K. Ana. Ausgang 0-10V
3   5555:2  PREOP  +  EL2004 4K. Dig. Ausgang 24V, 0,5A
```

7.1.18 Accès IDN SoE

```
ethercat soe_read [OPTIONS] <IDN>
ethercat soe_read [OPTIONS] <DRIVE> <IDN>

Read an SoE IDN from a slave.

This command requires a single slave to be selected.

Arguments:
  DRIVE    is the drive number (0 - 7). If omitted, 0 is assumed.
  IDN      is the IDN and must be either an unsigned
           16 bit number acc. to IEC 61800-7-204:
             Bit 15: (0) Standard data, (1) Product data
             Bit 14 - 12: Parameter set (0 - 7)
             Bit 11 - 0: Data block number
           or a string like 'P-0-150'.

Data of the given IDN are read and displayed according to
the given datatype, or as raw hex bytes.

These are valid data types to use with
the --type option:
  bool,
  int8, int16, int32, int64,
  uint8, uint16, uint32, uint64,
  float, double,
  string, octet_string, unicode_string.
For sign-and-magnitude coding, use the following types:
  sm8, sm16, sm32, sm64
```

```
Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --type     -t <type>   Data type (see above).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.


ethercat soe_write [OPTIONS] <IDN> <VALUE>
ethercat soe_write [OPTIONS] <DRIVE> <IDN> <VALUE>

Write an SoE IDN to a slave.

This command requires a single slave to be selected.

Arguments:
  DRIVE    is the drive number (0 - 7). If omitted, 0 is assumed.
  IDN      is the IDN and must be either an unsigned
           16 bit number acc. to IEC 61800-7-204:
             Bit 15: (0) Standard data, (1) Product data
             Bit 14 - 12: Parameter set (0 - 7)
             Bit 11 - 0: Data block number
           or a string like 'P-0-150'.
  VALUE    is the value to write (see below).

The VALUE argument is interpreted as the given data type
(--type is mandatory) and written to the selected slave.

These are valid data types to use with
the --type option:
  bool,
  int8, int16, int32, int64,
  uint8, uint16, uint32, uint64,
  float, double,
  string, octet_string, unicode_string.
For sign-and-magnitude coding, use the following types:
  sm8, sm16, sm32, sm64

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.
  --type     -t <type>   Data type (see above).

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.19 Demande des états de la couche application

```
ethercat states [OPTIONS] <STATE>

Request application-layer states.

Arguments:
  STATE can be 'INIT', 'PREOP', 'BOOT', 'SAFEOP', or 'OP'.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.1.20 Affichage de la version du maître

```
ethercat version [OPTIONS]

Show version information.
```

7.1.21 Génération de la description de l'esclave au format XML

```
ethercat xml [OPTIONS]

Generate slave information XML.

Note that the PDO information can either originate
from the SII or from the CoE communication area. For
slaves, that support configuring PDO assignment and
mapping, the output depends on the last configuration.

Command-specific options:
  --alias    -a <alias>
  --position -p <pos>    Slave selection. See the help of
                         the 'slaves' command.

Numerical values can be specified either with decimal (no
prefix), octal (prefix '0') or hexadecimal (prefix '0x') base.
```

7.2 Bibliothèque en espace utilisateur

L'API native (voir chapitre 3) se trouve dans l'espace noyau et n'est donc accessible que depuis le noyau. Pour rendre l'API disponible aux programmes en espace utilisateur, une bibliothèque en espace utilisateur a été créée, et elle peut être liée à des programmes selon les termes et conditions de la licence LGPL, version 2 [5].

La bibliothèque s'appelle *libethercat*. Ses sources se trouvent dans le sous-dossier *lib/* et elles sont construites par défaut lorsqu'on utilise la commande `make`. Elle est installée

dans le sous-dossier *lib/* en dessous du préfixe d'installation sous le nom *libethercat.a* (pour la liaison statique), *libethercat.la* (pour utiliser avec *libtool*) et *libethercat.so* (pour la liaison dynamique).

7.2.1 Utilisation de la bibliothèque

Le fichier d'entête *ecrt.h* de l'API peut être utilisé dans les deux contextes : utilisateur ou noyau.

L'exemple minimal suivant montre comment construire un programme EtherCAT. Un exemple complet se trouve dans le dossier *examples/user/* des sources du maître.

```
#include <ecrt.h>

int main(void)
{
    ec_master_t *master = ecrt_request_master(0);

    if (!master)
        return 1; // error

    pause(); // wait for signal
    return 0;
}
```

Le programme peut être compilé et dynamiquement lié à la bibliothèque avec la commande ci-dessous :

Listing 7.1 – Commande de l'éditeur de liens pour utiliser la bibliothèque de l'espace utilisateur

```
gcc ethercat.c -o ectest -I/opt/etherlab/include \
    -L/opt/etherlab/lib -lethercat \
    -Wl,--rpath -Wl,/opt/etherlab/lib
```

La bibliothèque peut aussi être liée statiquement au programme :

```
gcc -static ectest.c -o ectest -I/opt/etherlab/include \
    /opt/etherlab/lib/libethercat.a
```

7.2.2 Implémentation

Fondamentalement, l'API noyau a été transferée dans l'espace utilisateur via le périphérique en mode caractére du maître (voir chapitre 2, figure 2.1 et sous-section 7.1.1).

Les appels de fonction de l'API noyau sont projetés dans l'espace utilisateur via l'interface `ioctl()`. Les fonctions de l'API en espace utilisateur partagent un ensemble

d'appels `ioctl()` génériques. La partie noyau des appels de l'interface appelle directement les fonctions correspondantes de l'API, ce qui ajoute un minimum de délai supplémentaire (voir sous-section 7.2.3).
Pour des raisons de performance, les données de processus réels (voir section 2.3) ne sont pas copiées entre la mémoire du noyau et celle de l'utilisateur : à la place, les données sont projetées en mémoire vers l'application en espace utilisateur. Une fois que le maître est configuré et activé, le module maître crée une zone de mémoire de données de processus couvrant tous les domaines et la mappe dans l'espace utilisateur, de sorte que l'application puisse accéder directement aux données de processus. En conséquence, il n'y a pas de délai supplémentaire lors de l'accès aux données de processus depuis l'espace utilisateur.

Différence API noyau/utilisateur En raison de la projection en mémoire des données de processus, la mémoire est gérée en interne par les fonctions de la bibliothèque. Par conséquent, il est impossible de fournir de la mémoire externe pour les domaines, comme pour l'API noyau. Les fonctions correspondantes sont disponibles uniquement dans l'espace noyau. C'est la seule différence lorsqu'on utilise l'API depuis l'espace utilisateur.

7.2.3 Timing

Un aspect intéressant est la comparaison du timing des appels de la bibliothèque en espace utilisateur avec ceux de l'API noyau. tableau 7.1 montre les durées des appels et l'écart-type des fonctions de l'API typiques (et critiques pour le temps) mesurée avec un processeur Intel Pentium 4 M avec 2.2 GHz et un noyau standard 2.6.26.

TABLE 7.1 – Comparaison du timing des API

	Espace noyau		**Espace utilisateur**	
Fonction	$\mu(t)$	$\sigma(t)$	$\mu(t)$	$\sigma(t)$
`ecrt_master_receive()`	1.1 µs	0.3 µs	2.2 µs	0.5 µs
`ecrt_domain_process()`	< 0.1 µs	< 0.1 µs	1.0 µs	0.2 µs
`ecrt_domain_queue()`	< 0.1 µs	< 0.1 µs	1.0 µs	0.1 µs
`ecrt_master_send()`	1.8 µs	0.2 µs	2.5 µs	0.5 µs

Les résultats des tests montrent que, dans cette configuration, l'API en espace utilisateur rajoute un délai supplémentaire d'environ 1 µs à chaque fonction, par rapport à l'API en mode noyau.

7.3 Interface RTDM

Lorsqu'on utilise les interfaces en espace utilisateur des extensions temps réels telles que Xenomai ou RTAI, il est déconseillé d'utiliser *ioctl()*, parce que ça peut perturber

les opérations en temps réels. Pour y parvenir, le modèle de périphérique temps réel (Real-Time Device Model = RTDM[17]) a été développé. Le module maître fourni une interface RTDM (voir figure 2.1) en plus du périphérique normal en mode caractère, si les sources du maîtres sont configurées avec `--enable-rtdm` (voir chapitre 9).

Pour forcer une application à utiliser l'interface RTDM au lieu du périphérique normal en mode caractères, elle doit être liée avec la bibliothèque *libethercat_rtdm* au lieu de *libethercat*. L'utilisation de *libethercat_rtdm* est transparente, par conséquent l'entête EtherCAT *ecrt.h* peut être utilisé comme d'habitude avec l'API complète.

Pour construire l'exemple dans Listing 7.1 avec la bibliothèque RTDM, la commande de l'éditeur de lien doit être modifiée comme ci-dessous :

```
gcc ethercat-with-rtdm.c -o ectest -I/opt/etherlab/include \
    -L/opt/etherlab/lib -lethercat_rtdm \
    -Wl,--rpath -Wl,/opt/etherlab/lib
```

7.4 Intégration système

Pour intégrer le maître EtherCAT en tant que service dans un système en cours d'exécution, il vient avec un script d'initialisation et un fichier sysconfig qui sont décrits ci-dessous. Les systèmes plus modernes utilisent systemd [7]. L'intégration du maître avec systemd est décrite dans sous-section 7.4.4.

7.4.1 Script d'initialisation

Le script d'initialisation du maître EtherCAT est conforme aux exigences de la "Linux Standard Base" ((LSB, [6])). Le script est installé dans *etc/init.d/ethercat* sous le préfixe d'installation et doit être copié (ou encore mieux : lié) aux destinations appropriées (voirchapitre 9), avant que le maître puisse être inséré en tant que service. Veuillez noter, que ce script d'initialisation dépend du fichier sysconfig décrit ci-dessous.

Pour indiquer les dépendances du service (c'est-à-dire, quels services doivent être démarrés avant les autres) à l'intérieur du code du script d'initialisation, LSB définit un bloc spécial de commentaires. Les outils systèmes peuvent extraire cette information pour insérer le script d'initialisation EtherCAT à la bonne position dans la séquence de démarrage :

```
# Default-Stop:      0 1 2 6
# Short-Description: EtherCAT master
# Description:       EtherCAT master @VERSION@
### END INIT INFO

#------------------------------------------------------------------------------

ETHERCATCTL="@sbindir@/ethercatctl -c @sysconfdir@/sysconfig/ethercat"

#------------------------------------------------------------------------------
```

7.4.2 Fichier sysconfig

Pour la configuration persistante, le script d'initialisation utilise un fichier sysconfig installé dans *etc/sysconfig/ethercat* (sous le préfixe d'installation), qui est obligatoire. Le fichier sysconfig contient toutes les variables de configuration requises pour opérer un ou plusieurs maîtres. La documentation se trouve dans le fichier et elle est reproduite ci-dessous :

```
#
# The MASTER<X>_DEVICE variable specifies the Ethernet device for a master
# with index 'X'.
#
# Specify the MAC address (hexadecimal with colons) of the Ethernet device to
# use. Example: "00:00:08:44:ab:66"
#
# Alternatively, a network interface name can be specified. The interface
# name will be resolved to a MAC address using the 'ip' command.
# Example: "eth0"
#
# The broadcast address "ff:ff:ff:ff:ff:ff" has a special meaning: It tells
# the master to accept the first device offered by any Ethernet driver.
#
# The MASTER<X>_DEVICE variables also determine, how many masters will be
# created: A non-empty variable MASTER0_DEVICE will create one master, adding a
# non-empty variable MASTER1_DEVICE will create a second master, and so on.
#
# Examples:
# MASTER0_DEVICE="00:00:08:44:ab:66"
# MASTER0_DEVICE="eth0"
#
MASTER0_DEVICE=""
#MASTER1_DEVICE=""

#
# Backup Ethernet devices
#
# The MASTER<X>_BACKUP variables specify the devices used for redundancy. They
# behaves nearly the same as the MASTER<X>_DEVICE variable, except that it
# does not interpret the ff:ff:ff:ff:ff:ff address.
#
#MASTER0_BACKUP=""

#
# Ethernet driver modules to use for EtherCAT operation.
#
# Specify a non-empty list of Ethernet drivers, that shall be used for
# EtherCAT operation.
#
# Except for the generic Ethernet driver module, the init script will try to
# unload the usual Ethernet driver modules in the list and replace them with
# the EtherCAT-capable ones. If a certain (EtherCAT-capable) driver is not
# found, a warning will appear.
#
# Possible values: 8139too, e100, e1000, e1000e, r8169, generic, ccat, igb, igc,
    genet, dwmac-intel, stmmac-pci.
# Separate multiple drivers with spaces.
# A list of all matching kernel versions can be found here:
# https://docs.etherlab.org/ethercat/1.6/doxygen/devicedrivers.html
#
# Note: The e100, e1000, e1000e, r8169, ccat, igb and igc drivers are not built by
# default. Enable them with the --enable-<driver> configure switches.
#
DEVICE_MODULES=""
```

```
55
56 # If you have any issues about network interfaces not being configured
57 # properly, systemd may need some additional infos about your setup.
58 # Have a look at the service file, you'll find some details there.
59 #
60
61 #
62 # List of interfaces to bring up and down automatically.
63 #
64 # Specify a space-separated list of interface names (such as eth0 or
65 # enp0s1) that shall be brought up on `ethercatctl start` and down on
66 # `ethercatctl stop`.
67 #
68 # When using the generic driver, the corresponding Ethernet device has to be
69 # activated before the master is started, otherwise all frames will time out.
70 # This the perfect use-case for `UPDOWN_INTERFACES`.
71 #
72 UPDOWN_INTERFACES=""
73
74 #
75 # Flags for loading kernel modules.
76 #
77 # This can usually be left empty. Adjust this variable, if you have problems
78 # with module loading.
79 #
80 #MODPROBE_FLAGS="-b"
81
82 #------------------------------------------------------------------------------
```

Pour les systèmes gérés par systemd (voir sous-section 7.4.4), le fichier sysconfig a été déplacé dans `/etc/ethercat.conf`. Les deux versions font parties des sources du maître et sont destinées à être utilisées en alternance.

7.4.3 Démarrage du maître comme service

Une fois que le script d'initialisation et le fichier sysconfig ont été installés au bon endroit, le maître EtherCAT peut être inséré comme un service. Les différentes distributions Linux offrent différentes façons pour marquer un service pour le démarrage ou l'arrêt dans certains runlevels. Par exemple, SUSE Linux fournit la commande *insserv* :

```
# insserv ethercat
```

Le script d'initialisation peut aussi être utilisé pour démarrer ou stopper manuellement le maître EtherCAT.

Il doit être exécuté avec un des paramètres suivants : `start`, `stop`, `restart` ou `status`.

```
# /etc/init.d/ethercat restart
Shutting down EtherCAT master                    done
Starting EtherCAT master                         done
```

7.4.4 Intégration avec systemd

Les distributions utilisant *systemd* à la place du système d'initialisation SysV utilisent des fichiers de service pour décrire comment un service doit être maintenu. Listing 7.2

liste le fichier de service du maître :

Listing 7.2 – Service file

```
#
# EtherCAT master kernel modules
#

[Unit]
Description=EtherCAT Master Kernel Modules

# Fine tuning of the startup dependencies below are recommended
# to provide a reliable startup routine.
# The dependencies below can be either uncommented after copying
# this file to /etc/systemd/system or by creating overrides:
# Copy the needed dependencies into
# /etc/systemd/system/ethercat.service.d/50-dependencies.conf
# in a [Unit] section.

#
# Uncomment this, if the generic Ethernet driver is used. It assures, that the
# network interfaces are configured, before the master starts.
#
#Requires=network.target # Stop master, if network is stopped
#After=network.target # Start master, after network is ready

#
# Uncomment this, if a native Ethernet driver is used. It assures, that the
# network interfaces are configured, after the Ethernet drivers have been
# replaced. Otherwise, the networking configuration tools could be confused.
#
#Before=network-pre.target
#Wants=network-pre.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=@sbindir@/ethercatctl start
ExecStop=@sbindir@/ethercatctl stop

[Install]
WantedBy=multi-user.target
```

La commande *systemctl* est utilisée pour charger et décharger le maître et les modules des pilotes réseaux de la même manière que l'ancien script d'initialisation (sous-section 7.4.1).

```
# systemctl start ethercat
```

Lorsqu'on utilise systemd et/ou la commande *systemctl*, le fichier de configuration du maître doit être dans `/etc/ethercat.conf` au lieu de `/etc/sysconfig/ethercat` ! Celui-ci est ignoré. Les options de configurations sont exactement les mêmes.

7.5 Interfaces de déverminage

Les bus EtherCAT peuvent toujours être surveillés en insérant un commutateur entre le maître et l'esclave. Ceci permet de connecter un autre PC avec un analyseur réseau,

par exemple Wireshark [8]. Il est aussi possible d'écouter directement sur les interfaces réseaux locales de la machine exécutant le maître EtherCAT. Si le pilote Ethernet générique (voir section 4.3) est utilisé, l'analyseur réseau peut écouter directement sur l'interface réseau connecté au bus EtherCAT.

Si on utilise les pilotes Ethernet natifs (voir section 4.2), il n'y a aucune interface réseau local pour écouter, parce que les périphériques Ethernet utilisés pour EtherCAT ne sont par enregistrés dans la pile réseau. Dans ce cas, des "interfaces de déverminage" sont supportées : ce sont des interfaces réseaux virtuelles pour permettre la capture du trafic EtherCAT avec un analyseur réseau (comme Wireshark ou tcpdump) s'exécutant sur la machine maîtresse sans utiliser de matériel externe. Pour utiliser cette fonctionnalité, les sources du maître doivent avoir été configurées avec l'option `--enable-debug-if` (voir chapitre 9).

Chaque maître EtherCAT enregistre une interface réseau en lecture seule par périphérique Ethernet physique. Les interfaces réseaux sont nommées *ecdbgmX* pour le périphérique principal et *ecdbgbX* pour le périphérique de secours, où X est l'index du maître. Le listing ci-dessous montre une interface de déverminage parmi des interfaces réseaux standards :

```
# ip link
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop qlen 1000
    link/ether 00:13:46:3b:ad:d7 brd ff:ff:ff:ff:ff:ff
8: ecdbgm0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast
                                                qlen 1000
    link/ether 00:04:61:03:d1:01 brd ff:ff:ff:ff:ff:ff
```

Lorsque l'interface de déverminage est activée, toutes les trames envoyées ou reçues depuis ou vers le périphérique physique sont aussi transmises à l'interface de déverminage par le maître correspondant. Les interfaces réseaux peuvent être activées avec la commande ci-dessous :

```
# ip link set dev ecdbgm0 up
```

Veuillez noter, que la fréquence des trames peut être très élevée. Avec une application connectée, l'interface de déverminage peut produire des milliers de trames par seconde.

Attention Les tampons de socket nécessaires pour les interfaces de déverminage doivent être alloués dynamiquement. Certaines extensions temps réels pour Linux (comme RTAI) ne l'autorisent pas un contexte temps réel !

8 Aspects temporels

Bien que le timing EtherCAT soit hautement déterministe et que par conséquent les problèmes de timing soient rares, il y a quelques aspects qui peuvent (et doivent) être traités.

8.1 Profilage de l'interface de programmation applicative

Un des aspects de timing les plus important est le temps d'exécution des fonctions de l'API, qui sont appelées dans un contexte cyclique. Ces fonctions prennent une part importante du timing d'ensemble de l'application. Pour mesurer le timing de ces fonctions, le code suivant a été utilisé :

```
c0 = get_cycles();
ecrt_master_receive(master);
c1 = get_cycles();
ecrt_domain_process(domain1);
c2 = get_cycles();
ecrt_master_run(master);
c3 = get_cycles();
ecrt_master_send(master);
c4 = get_cycles();
```

Entre chaque appel d'une fonction de l'API, le compteur d'horodatage d'estampille du microprocesseur est lu. Les différences des compteurs sont converties en µs au moyen de la variable `cpu_khz`, qui contient le nombre d'incréments par ms.

Pour la mesure réelle, un système avec un microprocesseur à 2.0 GHz a été utilisé pour exécuter le code ci-dessus dans un fil d'exécution RTAI avec une période de 100 µs. La mesure a été répétée $n = 100$ fois et les résultats ont été moyennés. Ils sont visibles dans tableau 8.1.

Il est évident, que les fonctions qui accèdent au matériel prennent la part du lion. La fonction *ec_master_receive()* exécute la requête de service d'interruption (ISR) du périphérique Ethernet, analyse les datagrammes et copie leurs contenus dans la mémoire des objets datagrammes. La fonction *ec_master_send()* assemble une trame à partir des datagrammes et la copie vers les tampons matériels. Il est intéressant de noter, que ceci ne prend qu'un quart du temps de réception.

TABLE 8.1 – Profilage d'un cycle d'application sur un processeur à 2.0 GHz

Élement	Durée moyenne [s]	Déviation standard [µs]
ecrt_master_receive()	8.04	0.48
ecrt_domain_process()	0.14	0.03
ecrt_master_run()	0.29	0.12
ecrt_master_send()	2.18	0.17
Cycle complet	10.65	0.69

Les fonctions qui opèrent uniquement sur les structures de données internes des maîtres sont très rapides ($\Delta t < 1$ µs). Il est intéressant de noter que l'exécution de *ec_domain_process()* a un petit écart-type par rapport à la moyenne, alors que le ratio est presque deux fois plus grand pour *ec_master_run()* : Cela vient probablement des fonctions ultérieures qui doivent exécuter le code en fonction de l'état courant et les différentes fonctions d'état sont plus ou moins complexes.

Pour un cycle en temps réel qui représente environ 10 µs, la fréquence théorique peut atteindre jusqu'à 100 kHz. Mais cette fréquence reste théorique pour deux raisons :

1. Le processeur doit continuer à exécuter le système d'exploitation entre les cycles temps réels.
2. Les trames EtherCAT doivent être envoyées et reçues, avant que le prochain cycle temps réel commence. La détermination du temps de cycle du bus est difficile. Elle est couverte dans section 8.2.

8.2 Mesure des cycles du bus

Pour mesurer le temps pendant lequel, la trame est "sur le câble", deux horodatages sont nécessaires :

1. Le premier quand le matériel Ethernet commence à envoyer physiquement la trame.
2. Le second quand la trame est complètement reçue par le matériel Ethernet.

Les deux instants sont difficiles à déterminer. La première raison est que les interruptions sont désactivées et le maître n'est pas notifié quand une trame est envoyée ou reçue (un sondage fausserait les résultats). La deuxième raison est que, même avec les interruptions activées, la durée entre l'évènement et la notification est inconnue. C'est pourquoi, la seule manière de déterminer avec certitude le temps de cycle du bus est une mesure électrique.

De toute façon, la durée du cycle du bus est un facteur important lors de la conception du code temps réel, car il limite la fréquence maximale pour la tâche cyclique de l'application. En pratique, ces paramètres de timing dépendent fortement du matériel

et une méthode par essais et erreurs doit être utilisée pour déterminer les limites du système.

La question centrale est : Que se passe-t-il si la fréquence du cycle est trop haute ? La réponse est que les trames EtherCAT qui ont été envoyées à la fin du cycle ne sont pas encore reçues quand le prochain cycle démarre.

Ceci est notifié en premier par *ecrt_domain_process()*, parce que le compteur de travail des datagrammes de données de processus n'est pas incrémenté. La fonction notifiera l'utilisateur via Syslog [1]. Dans ce cas, les données de processus sont conservés identiques comme dans le dernier cycle, parce qu'elles ne sont pas écrasées par le domaine. Quand les datagrammes du domaine sont à nouveau mis en file d'attente, le maître s'aperçoit qu'ils ont déjà été mis en file d'attente (et marqués comme envoyés). Le maître les marquera à nouveau comme non-envoyés et affichera un avertissement que les datagrammes ont été "ignorés".

Sur le système à 2.0 GHz mentionné, la fréquence de cycle possible peut atteindre 25 kHz sans perdre de trames. Cette valeur peut sûrement être augmentée en choisissant un matériel plus rapide. En particulier le matériel réseau RealTek peut être remplacé par un autre plus rapide. En outre, la mise en oeuvre d'un ISR dédié pour les périphériques EtherCAT contribuerait également à augmenter la latence. Ces deux points sont la liste des choses encore à faire de l'auteur.

1. Pour limiter la sortie de Syslog, un mécanisme a été implémenté pour générer une notification résumée au maximum une fois par seconde.

9 Installation

9.1 Obtention du logiciel

Il y a plusieurs manières d'obtenir le logiciel du maître :

1. Une version officielle (par exemple 1.5.2) peut être téléchargée depuis le site web du maître [1] dans le projet EtherLab [1] sous forme d'archive tar.
2. La révision de développement la plus récente (mais aussi n'importe quelle autre révision) peut être obtenue via le dépôt Git [14] sur la page du projet sur GitLab.com [2]. L'intégralité du dépot peut être clonée avec la commande

   ```
   git clone https://gitlab.com/etherlab.org/ethercat.git
       local-dir
   ```

3. Sans installation locale de Git, des archives tar de révisions arbitraires peuvent être téléchargées via le bouton "Download" sur GitLab.

9.2 Construction du logiciel

Après le téléchargement d'une archive tar ou le clonage du dépôt tel que décrit dans la section 9.1, les sources doivent être préparées et configurées pour le processus de construction.

Si une archive tar a été téléchargée, elle doit être extraite avec les commandes suivantes :

```
$ tar xjf ethercat-1.5.2.tar.bz2
$ cd ethercat-1.5.2/
```

La configuration du logiciel est gérée avec Autoconf [15] aussi les versions publiées contiennent un script shell `configure`, qui doit être exécuté pour la configuration (voir ci-dessous).

Amorcage Lors d'un téléchargement ou clonage direct du dépôt, le script `configure` n'existe pas encore. Il peut être créé via le script `bootstrap.sh` dans les sources du maître. Les paquets autoconf et automake sont alors nécessaires.

1. https://etherlab.org/ethercat
2. https://gitlab.com/etherlab.org/ethercat

Configuration et construction La configuration et le processus de construction suivent dans les commandes ci-dessous :

```
$ ./configure
$ make
$ make modules
```

tableau 9.1 liste les commutateurs importants de configuration et les options :

TABLE 9.1 – Options de configuration

Option/Commutateur	Description	Défaut
`--prefix`	Préfixe d'installation	*/opt/etherlab*
`--with-linux-dir`	Sources du noyau Linux	Utilise le noyau actuel
`--with-module-dir`	Sous-dossier dans l'arbre des modules du noyau dans lequel les modules noyaux EtherCAT doivent être installés.	*ethercat*
`--enable-generic`	Construire le pilote Ethernet générique (voir section 4.3).	oui
`--enable-8139too`	Construire le pilote 8139too	oui
`--with-8139too-kernel`	noyau 8139too	†
`--enable-e100`	Construire le pilote e100 driver	non
`--with-e100-kernel`	e100 noyau	†
`--enable-e1000`	Activer le pilote e1000	non
`--with-e1000-kernel`	noyau e1000	†
`--enable-e1000e`	Activer le pilote e1000e	non
`--with-e1000e-kernel`	noyau e1000e	†
`--enable-r8169`	Activer le pilote r8169	non
`--with-r8169-kernel`	noyau r8169	†
`--enable-ccat`	Activer le pilote ccat (indépendant de la version du noyau)	non
`--enable-igb`	Activer le pilote igb	non
`--with-igb-kernel`	noyau igb	†
`--enable-kernel`	Construire les modules noyau du maître	oui
`--enable-rtdm`	Créer l'interface RTDM (Le dossier RTAI ou Xenomai est requis, voir ci-dessous)	non
`--with-rtai-dir`	Chemin RTAI (pour les exemples RTAI et interface RTDM)	
`--with-xenomai-dir`	Chemin Xenomai (pour les exemples Xenomai et interface RTDM)	

Option/Commutateur	Description	Défaut
`--with-devices`	Nombre de périphériques Ethernet pour l'opération redondante (> 1 commute la redondance)	1
`--with-systemdsystemunitdir`	Chemin Systemd	auto
`--enable-debug-if`	Créer une interface de déverminage pour chaque maître	non
`--enable-debug-ring`	Créer un anneau de déverminage pour enregistrer les trames	non
`--enable-eoe`	Activer le support EoE	oui
`--enable-cycles`	Utiliser le compteur d'horodatage du processeur. Activez ceci sur les architectures Intel pour un meilleur calcul des timings.	non
`--enable-hrtimer`	Utiliser un minuteur haute-résolution pour laisser dormir l'automate du maître entre l'envoi des trames.	non
`--enable-regalias`	Lire les alias d'adresses depuis le registre	non
`--enable-tool`	Construire l'outil en ligne de commande "ethercat" (voir section 7.1)	oui
`--enable-userlib`	Construire la bibliothèque pour l'espace utilisateur	oui
`--enable-tty`	Construire le pilote TTY	non
`--enable-wildcards`	Activer *0xffffffff* pour être un jocker pour l'identifiant de fabricant et le code produit	non
`--enable-sii-assign`	Activer l'assignation de l'accès SII à la couche PDI pendant la configuration de l'esclave	non
`--enable-rt-syslog`	Activer les instructions syslog dans le contexte temps réel	yes

† Si cette option n'est pas spécifiée, la version du noyau à utiliser est extraite des sources du noyau Linux.

9.3 Construction de la documentation de l'interface

Le code source est documenté avec Doxygen [13]. Pour construire la documentation HTML, le logiciel the Doxygen doit être installé. La commande ci-dessous génère les documents dans le sous-dossier *doxygen-output* :

```
$ make doc
```

La documentation de l'interface peut être consultée en ouvrant avec un navigateur web le fichier *doxygen-output/html/index.html*. Les fonctions et structures de données de l'application sont couvertes par leur propre module "Application Interface".

9.4 Installation du logiciel

Les commandes ci-dessous doivent être entrées en tant que *root* : la première installe l'entête EtherCAT, le script d'initialisation, le fichier sysconfig et l'outil en espace utilisateur dans le chemin du préfixe. La deuxième installe les modules noyaux dans le dossier des modules du noyau. L'appel final à `depmod` est nécessaire pour inclure les modules noyaux dans le fichier *modules.dep* pour permettre de les utiliser avec la commande `modprobe`, qui se trouve dans le script d'initialisation.

```
# make install
# make modules_install
# depmod
```

Si le dossier de destination des modules noyaux ne se trouve dans */lib/modules*, un dossier de destination différent peut être spécifié avec la variable make `DESTDIR`. Par exemple :

```
# make DESTDIR=/vol/nfs/root modules_install
```

Ce commande installe les modules noyaux compilés dans */vol/nfs/root/lib/modules*, auquel on ajoute la version du noyau.

Maintenant le fichier de configuration `/etc/sysconfig/ethercat` (voir sous-section 7.4.2) ou */etc/ethercat.conf* si on utilise systemd, doit être personnalisé. La personnalisation minimale consiste à définir la variable `MASTER0_DEVICE` avec l'adresse MAC du périphérique Ethernet à utiliser (ou `ff:ff:ff:ff:ff:ff` pour utiliser le premier périphérique offert) et à sélectionner le(s) pilote(s) à charger via la variable `DEVICE_MODULES`.

Après que la définition de la configuration de base, le maître peut être démarré avec la commande ci-dessous :

```
# systemctl start ethercat
```

Lorsqu'on utilise init.d, la commande suivante peut être utilisée à la place :

```
# /etc/init.d/ethercat start
```

A partir de cet instant, l'opération du maître peut être obervée en consultant les messages Syslog, qui ressemblent à ceux qui sont ci-dessous. Si des esclaves EtherCAT sont connectés au périphérique du maître EtherCAT, les indicateurs d'activité devraient commencer à clignoter.

```
1  EtherCAT: Master driver 1.5.2
2  EtherCAT: 1 master waiting for devices.
3  EtherCAT Intel(R) PRO/1000 Network Driver - version 6.0.60-k2
4  Copyright (c) 1999-2005 Intel Corporation.
5  PCI: Found IRQ 12 for device 0000:01:01.0
6  PCI: Sharing IRQ 12 with 0000:00:1d.2
7  PCI: Sharing IRQ 12 with 0000:00:1f.1
8  EtherCAT: Accepting device 00:0E:0C:DA:A2:20 for master 0.
9  EtherCAT: Starting master thread.
10 ec_e1000: ec0: e1000_probe: Intel(R) PRO/1000 Network
11           Connection
12 ec_e1000: ec0: e1000_watchdog_task: NIC Link is Up 100 Mbps
13           Full Duplex
14 EtherCAT: Link state changed to UP.
15 EtherCAT: 7 slave(s) responding.
16 EtherCAT: Slave states: PREOP.
17 EtherCAT: Scanning bus.
18 EtherCAT: Bus scanning completed in 431 ms.
```

① – ② Le module maître est en train de charger , et un maître est initialisé.

③ – ⑧ Le pilote e1000 compatible EtherCAT est en train de charger. Le maître accepte le périphérique avec l'adresse `00:0E:0C:DA:A2:20`.

⑨ – ⑯ Le maître entre en phase de repos, démarre son automate et commence à analyser le bus.

9.5 Création automatique des nœuds de périphériques

L'outil en ligne de commande `ethercat` (voir section 7.1) communique avec le maître via le périphérique en mode caractère. Les nœuds de périphériques correspondants sont créés automatiquement, si le service udev est en cours de fonctionnement. Veuillez noter, que pour certaines distributions, le paquet `udev` n'est pas installé par défaut.

Les nœuds de périphériques seront créés avec le mode `0660` et le groupe `root` par défaut. Si des utilisateurs "normaux" doivent avoir un accès en lecture, un fichier de règle udev (par exemple */etc/udev/rules.d/99-EtherCAT.rules*) doit être créé avec le contenu suivant :

```
KERNEL=="EtherCAT[0-9]*", MODE="0664"
```

Après la création du fichier de règles udev et le redémarrage du maître EtherCAT avec `/etc/init.d/ethercat restart`, le nœud de périphérique est automatiquement créé avec les bons droits :

```
# ls -l /dev/EtherCAT0
crw-rw-r--  1 root root 252, 0 2008-09-03 16:19 /dev/EtherCAT0
```

Maintenant, l'outil `ethercat` peut être utilisé (voir section 7.1) par un utilisateur non-root.

Si les utilisateurs non-root doivent avoir l'accès en écriture, on peut utiliser la règle udev suivante à la place :

```
KERNEL=="EtherCAT[0-9]*", MODE="0664", GROUP="users"
```

Bibliographie

[1] Ingenieurgemeinschaft IgH : EtherLab – Open Source Toolkit for rapid realtime code generation under Linux with Simulink/RTW and EtherCAT technology. https://etherlab.org, 2024.

[2] IEC 61158-4-12 : Data-link Protocol Specification. International Electrotechnical Commission (IEC), 2005.

[3] IEC 61158-6-12 : Application Layer Protocol Specification. International Electrotechnical Commission (IEC), 2005.

[4] GNU General Public License, Version 2. http://www.gnu.org/licenses/gpl-2.0.html. October 15, 2008.

[5] GNU Lesser General Public License, Version 2.1. http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html. October 15, 2008.

[6] Linux Standard Base. http://www.linuxfoundation.org/en/LSB. August 9, 2006.

[7] systemd System and Service Manager http://freedesktop.org/wiki/Software/systemd. January 18, 2013.

[8] Wireshark. http://www.wireshark.org. 2008.

[9] *Hopcroft, J. E. / Ullman, J. D.* : Introduction to Automata Theory, Languages and Computation. Adison-Wesley, Reading, Mass. 1979.

[10] *Wagner, F. / Wolstenholme, P.* : State machine misunderstandings. In : IEE journal "Computing and Control Engineering", 2004.

[11] RTAI. The RealTime Application Interface for Linux from DIAPM. https://www.rtai.org, 2010.

[12] RT PREEMPT HOWTO. http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO, 2010.

[13] Doxygen. Source code documentation generator tool. http://www.stack.nl/~dimitri/doxygen, 2008.

[14] Git SCM. https://git-scm.com, 2021.

[15] Autoconf – GNU Project – Free Software Foundation (FSF). http://www.gnu.org/software/autoconf, 2010.

[16] IEC 61800-7-304 : Adjustable speed electrical power drive systems - Part 7-300 : Generic interface and use of profiles for power drive systems - Mapping of profiles to network technologies. International Electrotechnical Commission (IEC), 2007.

[17] *J. Kiszka* : The Real-Time Driver Model and First Applications. http://svn.gna.org/svn/xenomai/tags/v2.4.0/doc/nodist/pdf/RTDM-and-Applications.pdf, 2013.

Glossaire

ADEOS Adaptive Domain Environment for Operating Systems, page 1

CoE CANopen over EtherCAT, Mailbox Protocol, page 56

ecdev EtherCAT Device, page 32

EoE Ethernet over EtherCAT, Mailbox Protocol, page 53

FSM Finite State Machine, page 35

ISR Interrupt Service Routine, page 26

LSB Linux Standard Base, page 2

PCI Peripheral Component Interconnect, Bus informatique, page 28

RTAI Realtime Application Interface, page 1

Index