2013-2014

Rapport de projet – EI4 AGI

Maquette pour un TP de réseau de terrain Ethernet Industriel temps réel

Projet réalisé par :

CUINIER Léna
DARROU Damien
LAAOUINA Marouane

Projet encadré par :

M. LAHAYE Sébastien
M. LAGRANGE Sébastien

IstiA
École d'Ingénieur
de l'Université d'Angers

université
angers

Engagement de non plagiat

Je, soussigné(e)
déclare être pleinement conscient(e) que le plagiat de documents ou d'une partie d'un document publiée sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour écrire ce rapport ou mémoire.

Signé par l'étudiant(e) le …. / …. / …....

Cet engament de non plagiat doit être signé et joint à tous les rapports, dossiers, mémoires.

ISTIA
62 Avenue Notre-Dame du Lac
49000 Angers cedex
Tél. 02 44 68 75 00 | Fax 02 44 68 75 01

Remerciements

Plus par élan de reconnaissance que par conformité à la tradition, nous consacrons cette page à nos remerciements.

Nous voudrions remercier vivement Mr. Lhommeau, le responsable d'AGI-EI4 et l'ensemble du corps administratif et professoral qui contribue à notre formation au sein de l'école. Il veille au bon déroulement de nos études dans des conditions favorables permettant une formation de haute qualité.

Nos sincères mots de remerciements vont à l'égard de notre tuteur Mr. Lahaye, enseignant à l'ISTIA pour son aide précieuse, pour ses conseils, ses instructions et ses directives qui nous ont permis la mise en œuvre de notre projet.

Enfin, nous adressons nos chaleureux remerciements et nos meilleurs respects à toutes les personnes qui nous ont aidées en sacrifiant une grande partie de leur temps à répondre à nos questions. Encore une fois, merci.

Sommaire

Introduction

Le monde évolue très vite et les réseaux de terrain ne font pas exception à la règle. En 2003 est apparu un nouveau protocole de réseau de terrain : Ethernet Powerlink, dont les qualités sont aujourd'hui bien connues : temps réel, déterminisme, rapidité.

Ainsi, dans le cadre des projets EI4 de l'école d'ingénieur ISTIA, il a été décidé de créer une maquette Ethernet Powerlink destinée aux étudiants dont le but est de mettre en avant les avantages de ce protocole.

L'élaboration du projet s'est faite chronologiquement. Savoir configurer un réseau Powerlink est une première étape délicate mais qu'il convient d'apprendre et de comprendre. Nous avons donc mis l'accent sur cette partie du projet pour que les « futurs » étudiants puisent bien assimiler ces notions.

Ensuite est venue une phase de programmation, l'idée étant de créer un petit soft pour calculer le temps cycle et mettre en avant le côté déterministe du protocole. En supplément, il semblait intéressant de faire une comparaison directe avec un autre protocole de réseau de terrain connu et très utilisé : Modbus.

Maquette de TP Powerlink

1) Qu'est ce que Powerlink ?

a) Définition

Powerlink est un protocole **temps réel** et **déterministe** pour **Ethernet standard**, c'est un système de communication fiable qui offre une grande flexibilité. Ce protocole de réseau de terrain a été créé en 2001 par la société **B&R**. Grâce à son potentiel, Powerlink permet d'établir la communication entre automates et ordinateurs en soutenant l'interconnexion des niveaux de contrôle, des processus et des réseaux de terrain.

Image 1 : logo de l'Ethernet Powerlink

Powerlink est aujourd'hui devenu une technologie fiable et efficace qui permet de concevoir les innovations et les machines du futur. Elle permet une large utilisation et fait partie d'une norme de la **CEI** (Commission électronique internationale). Cela lui a permis de gagner la confiance des fabricants de technologies de contrôle, de capteurs et de robotique.

b) Principe

- Les collisions sont évitées grâce au **principe des jetons** ainsi qu'à un temps de communication prédéfinie par le **Managing node**.
- Le protocole est basé sur le principe Maître / Esclaves.
- La communication est cyclique et des messages de synchronisation sont régulièrement émis par le **Managing node**.
- Une bande passante est réservée pour les **données asynchrones**.
- Les nœuds sont adressés via un **node ID** basé sur 8-Bit.

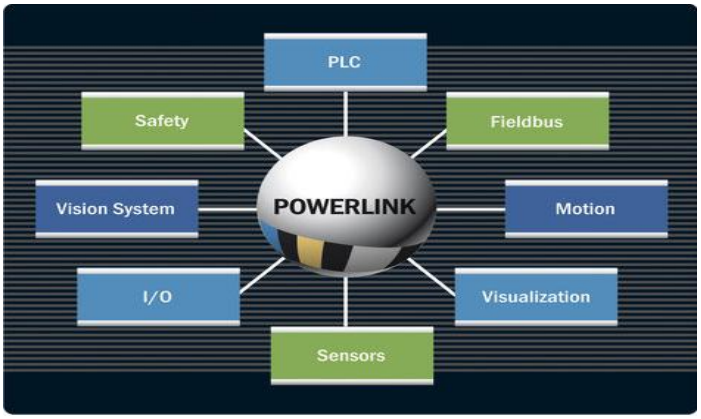


Image 2 : les différents champs d'action de l'Ethernet Powerlink

c) Modèle de communication

Le modèle de communication consiste à établir les échanges de données entre les équipements de manière cyclique, ce qui permet d'éviter les collisions et optimiser l'exploitation de la largeur de bande disponible. C'est à dire que le nœud gestionnaire (**Managing node**) pilote la communication tandis que les nœuds contrôlés (**Control node**) envoient les données demandées pendant leur temps de parole.

Le schéma suivant montre les étapes d'un cycle Powerlink :

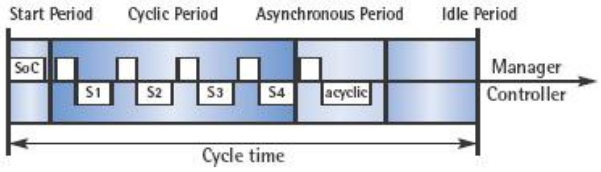


Image 3 : trame de communication Powerlink

Période de démarrage : Le Managing node envoie au début une trame « Start of Cyclic » à tous les Control node afin que les équipements du réseau soient synchronisés.

Période cyclique : Le Managing node envoie séquentiellement des « Poll Request » (requêtes) à chaque Control node, et celui contacté répond avec « Poll Response » (réponse). Les Control node reçoivent toutes les données et ne prennent que celles qui leur sont destinées.

Période asynchrone : Le Managing node attribue le droit d'émission à un Control node pour qu'il puisse transmettre une trame.

Période d'inactivité : Cette période correspond au temps restant jusqu'au commencement d'un nouveau cycle POWERLINK.

2) Objectif de la mission

Le sujet du projet est : **Maquette pour un TP de réseau de terrain Ethernet Industriel temps réel**. Après discussion avec notre superviseur, Mr LAHAYE Sébastien, le cahier des charges suivant a été établi.

a) Cahier des charges

Désireux de toujours proposer de nouveaux projets, l'ISTIA a décidé de créer une nouvelle maquette de TP pour les étudiants. La maquette à mettre en place concerne un nouveau protocole de réseau de terrain créé en 2003 : Powerlink.

Dans le cadre de fin de stage EI3, un étudiant est parti étudier le protocole Powerlink en Pologne pendant 3 mois pour y démontrer ses atouts majeurs : temps réel, rapidité et déterminisme. La fin de la mission s'est conclue par un succès : le protocole répond très bien à toutes ces attentes. Étant de plus en plus utilisé en milieu industriel, il est ici proposé de créer une maquette d'apprentissage à l'ISTIA.

Le but du projet est de mettre en place un TP implémentant cette technologie en vue de compléter le module intitulé « réseaux industriels » en EI4. Une base logicielle ainsi qu'un support écrit devront être créés pour permettre aux futurs étudiants de mettre le protocole en œuvre rapidement. La durée du TP sera d'environ 6 heures.

La maquette créée devra permettre aux futurs étudiants de comprendre les principes et les intérêts de ce nouveau protocole de réseau de terrain Powerlink via différentes phases :

- Mise en pratique PC – Automate reliée par un câble Powerlink.
- Simulation en temps réel pour saisir les enjeux du protocole (rapidité, fiabilité…).
- Création d'une seconde maquette utilisant un protocole moins récent pour comparer les données et prouver l'efficacité du protocole Powerlink.

b) Diagramme de Gantt

Suite à cela nous avons planifié un diagramme Gantt *(annexe 1)*. Dus aux différents problèmes rencontrés, nous n'avons pas pu suivre ce planning. Le planning suivi se trouve ci-dessous :

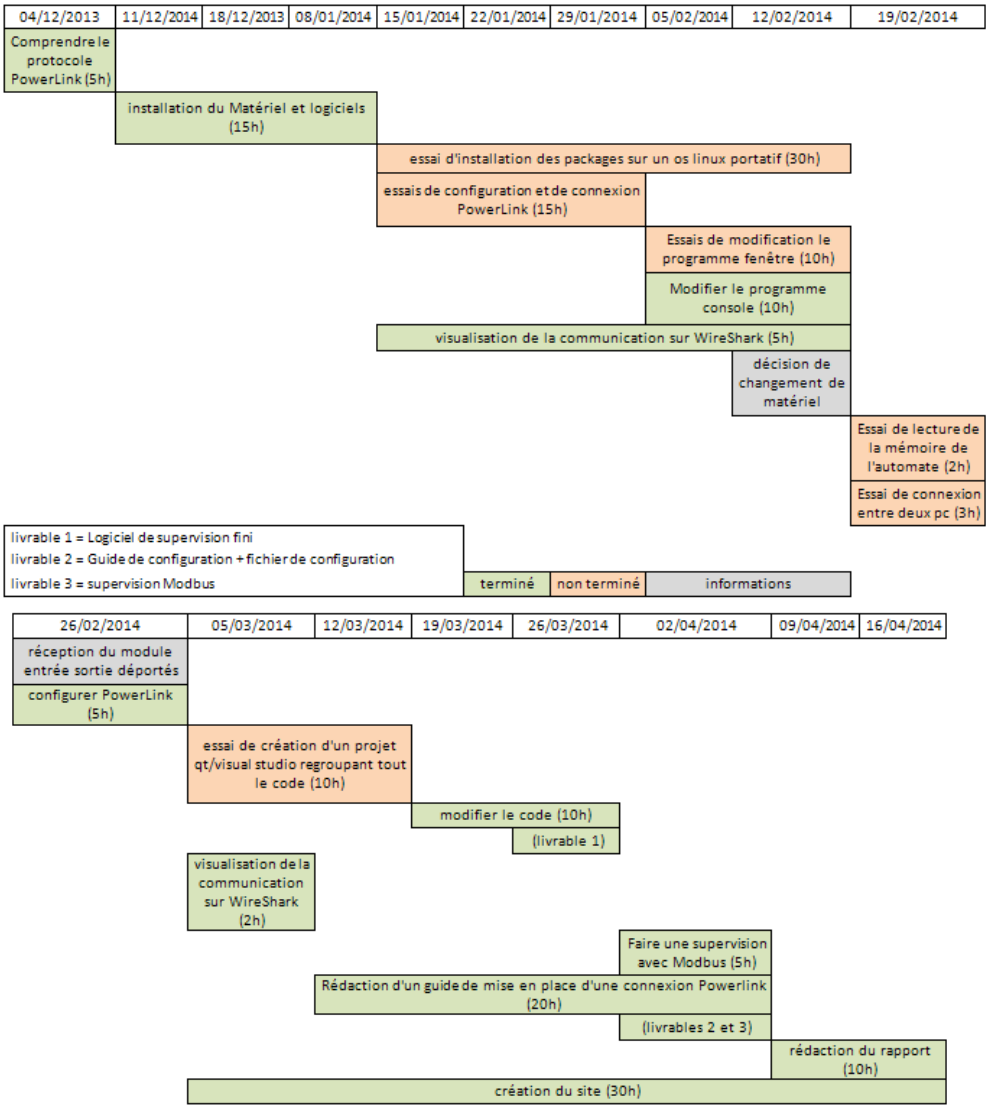


Image 4 : planning réel du projet

c) Répartition des taches

La répartition des tâches a été faite en fonction des préférences de chacun, certaines ont été faites en équipe.

Marouane Laaouina : Linux portatif, Création du site, Installation du matériel, Rédaction du rapport.

Damien Darou : Modification de code console et étude du code QT, Supervision Modbus, Essai de création d'un projet QT / Visual Studio, Création du site, Création des fichiers de configuration, Installation du matériel, Rédaction du rapport.

Léna Cuinier : Rédaction du guide openPowerlink, Modification du code console, Visualisation WireShark, Installation du matériel, Rédaction du rapport.

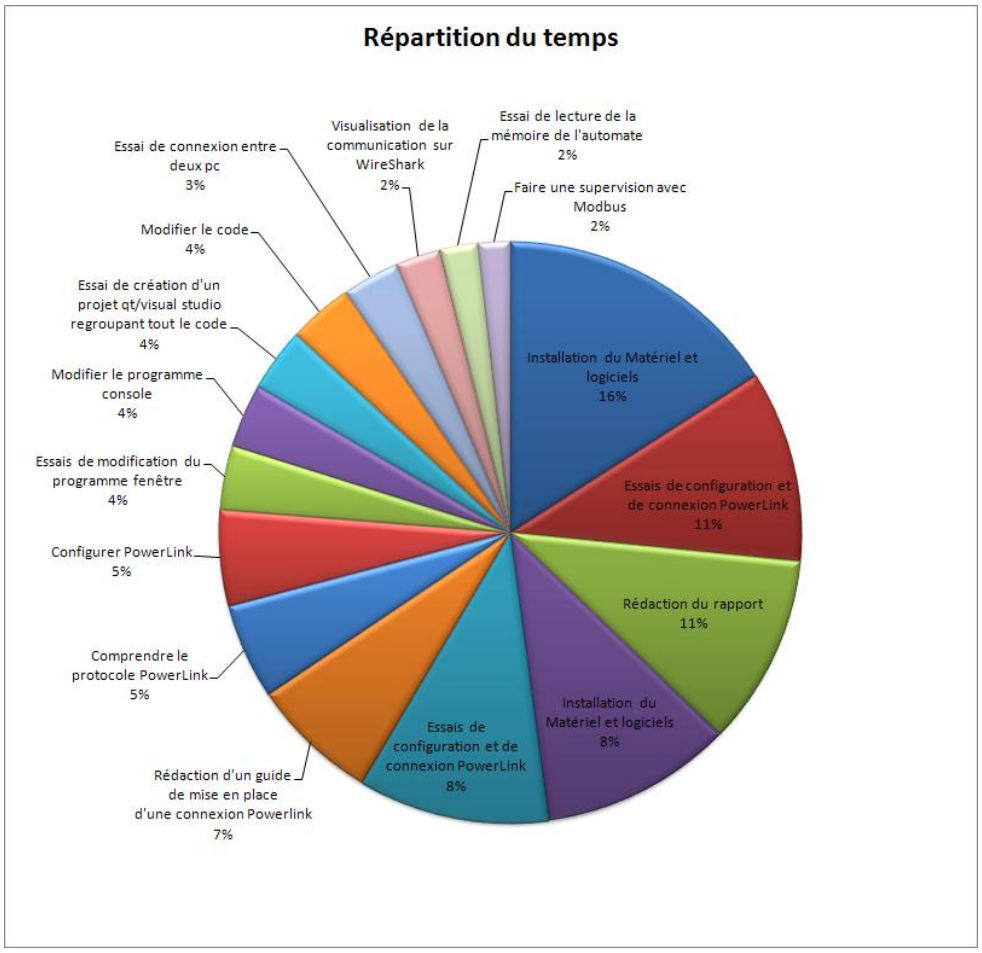


Image 5 : répartition des tâches

d) Situation initiale

Pour démarrer le projet, nous avions à disposition un automate **BR X20 CP 1484-1**, un module Powerlink **BR X20 IF 1082-2** et des documents issus de la présentation d'un intervenant du groupe BR. Cette documentation explique la procédure pour établir une connexion Powerlink entre un OS Ubuntu et un module **X20-BC0083** relié à des entrées sorties déportées.

Image 6 : l'automate BR C20 CP 1484-1

3) Procédure de création communication Powerlink

Pour répondre au cahier des charges, nous avons d'abord cherché à établir une communication Powerlink. Dans le but de simplifier cette tache dans le futur, nous avons regroupé toutes les actions nécessaires dans un guide (**guide openPowerlink**). La suite de ce chapitre résume ce guide.

Pour établir une connexion Powerlink, nous avons utilisé le logiciel **openPowerlink**. C'est un outil open source mis au point par B&R et d'autres fabriquant d'automates. Il permet d'établir une connexion entre un ordinateur tournant sur Linux et un équipement compatible avec le protocole Powerlink.

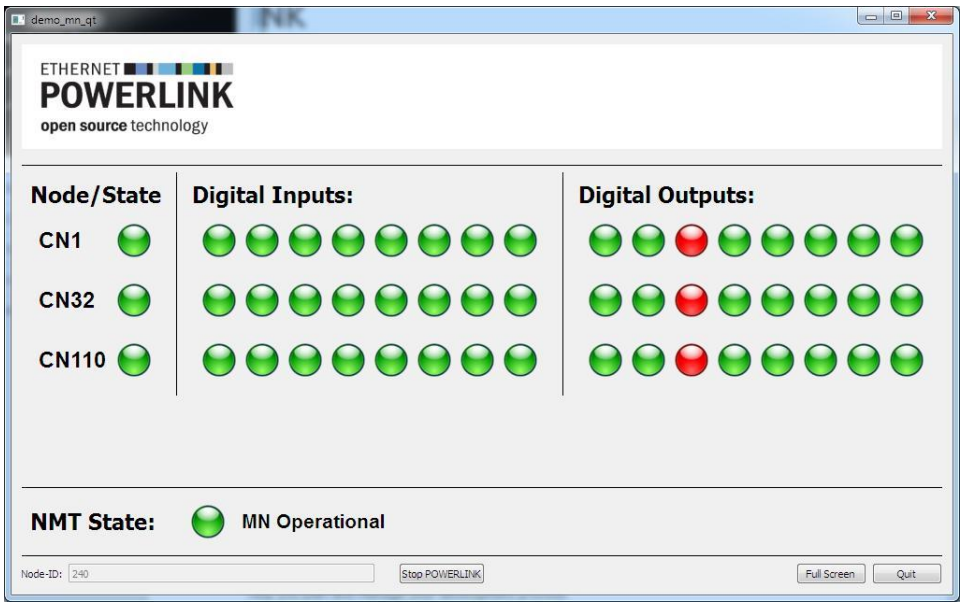


Image 7 : exemple openPowerlink avec 3 Control node

a) Prérequis

Le logiciel **openPowerlink** utilise les fichiers de configurations : **mnobd.cdc** et **xap.h**. Ils contiennent les informations relatives au Managing node et aux Control node. Ils permettent, entre autre, de définir les entrées / sorties des Control node et les trames pouvant être envoyées. La première étape est donc de créer ces fichiers.

FieldBus Designer permet de définir la composition d'entrées / sorties déportées et de les stoker dans un fichier .XDC. Pour ce faire, il suffit de configurer le module Powerlink ainsi que les module d'entrées / sorties utilisés pour générer le fichier .XDC.

Image 8 : logo des logiciels B&R

Ce fichier .XDC est ensuite utilisé à travers le logiciel **openConfigurator** pour obtenir les fichiers de configurations citées au-dessus : **mnobd.cdc** et **xap.h**.

On remarquera que le logiciel **openConfigurator** permet la modification du **temps de cycle** (Cycle Time). En effet ce temps doit être cohérent avec la carte Ethernet de l'ordinateur. La plupart des cartes Ethernet ont des difficultés en dessous de 10ms.

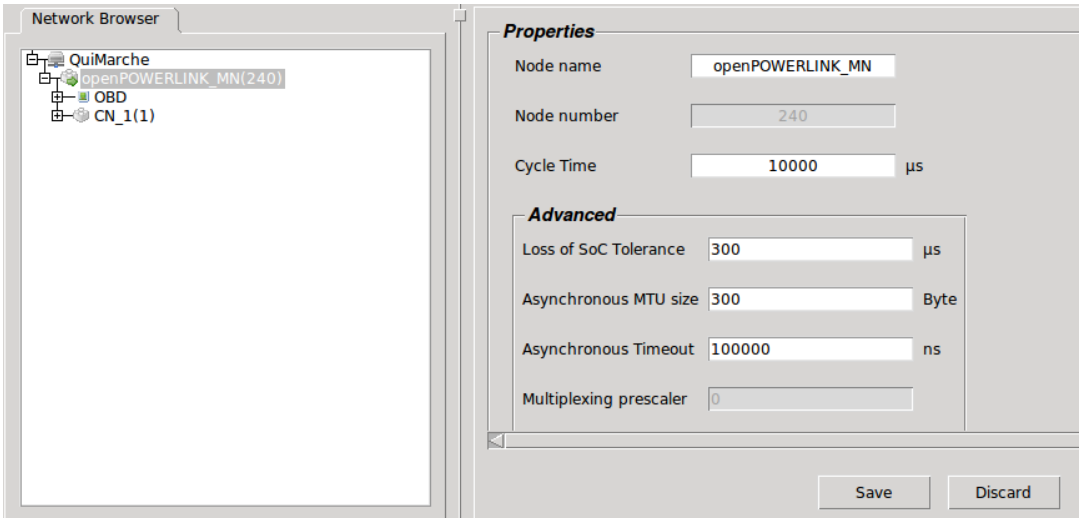


Image 9 : openConfigurator

Le fonctionnement du logiciel **openPowerlink** nécessite aussi l'installation de plusieurs librairies telles que cmake, g++, qt4-dev-tools ou libpcap-dev.

b) Téléchargement d'openPowerlink et mise en route

openPowerlink est téléchargeable sur le site : *http://sourceforge.net*.

Le dossier que nous obtenons après extraction contient du code. Les fichiers de configuration **mnobd.cdc** et **xap.h** doivent être inclus au niveau du code source des exécutables qui nous intéressent. Il faut ensuite compiler les **Makefile** pour obtenir ces derniers.

Les exécutables que nous avons utilisés sont « **demo_mn_console** » et « **demo_mn_qt** » contenus dans la version 1.08.4 d'openPowerlink.

« **demo_mn_qt** » permet l'affichage sous forme d'une fenêtre de l'état des Control node (déconnecté, pré-connecté, connecté) ainsi que l'état de leurs entrées / sorties (1 ou 0).

« **demo_mn_console** » permet l'affichage sous forme d'une console de l'état des Control node ainsi que l'exécution d'un programme facilement personnalisable.

On remarque que le code à modifier se trouve dans le fichier :
openPOWERLINK-V1.08.4/Examples/X86/Generic/demo_mn_console/demo_main.c

Il faut seulement modifier la fonction : *AppCbSync(void)* , située à la ligne 1017. Cette fonction s'exécute à chaque temps de cycle. Le programme que nous avons utilisé est détaillé *chapitre II.3*, la démo initial est étudiée *chapitre II.2*.

Développement du projet

1) Outils utilisés

a) openConfigurator

OpenConfigurator est un outil de configuration open-source qui permet de faciliter la configuration et la maintenance d'un réseau Powerlink.

- Implémenté en C++.
- Génère le fichier .CDC du maître openPowerlink avec ses Control node et permet la configuration automatique de l'ensemble du réseau Powerlink.
- Génère les variables de réseau utilisées (entrées / sorties des Control node) pour l'intégration avec le réseau Powerlink (fichier **xap.h**).
- Fonctionne sur Linux et Windows.

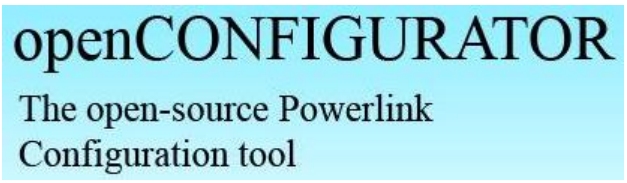


Image 10 : logo du logiciel openConfigurator

b) CMake

CMake est un logiciel destiné à compiler un projet développé sur plusieurs plateformes, par exemple sous Linux en compilant les fichiers Makefile. A l'inverse, sous Windows, on peut également utiliser des environnements de développement comme Visual Studio, Qt Creator. Pour remédier à cette problématique, CMake génère une solution de compilation pour le système cible.

Nous avons utilisé le logiciel **CMake-gui** pour configurer l'exécution du Makefile situé à la base du dossier openPowerlink. En effet, il est possible de choisir le dossier de sortie ainsi que les exécutables à générer.

c) Systèmes d'exploitation

Nous avons utilisé l'OS Windows lors de la création du fichier .XDC car le logiciel **FieldBus Designer** ne peut fonctionner sur un autre environnement. Cependant, nous avons très rapidement basculé sur Linux car le logiciel **openPowerlink** est difficile d'utilisation sur Windows. En effet, cela nécessitait la compilation de Makefile (effectuée facilement et rapidement grâce à la commande "make" sous Unix).

L'objectif de base était de faire fonctionner le TP sur une clé Linux mais ça n'a pas abouti à cause d'un problème lors du téléchargement des packages essentiels pour le fonctionnement d'openPowerlink, impossible à installer sur une clé.

Il semblerait que la solution idéale soit d'installer un linux virtuel sur une machine Windows. Il est ainsi plus facile de passer d'un système d'exploitation à l'autre : Windows pour la configuration XDC avec **Fieldbus Designer**, Linux pour la configuration d'openPowerlink.

d) Matériel et documentation fournie

Nous a été fourni lors du démarrage du projet :

- Un automate BR (**X20 CP 1484-1**) compatible Powerlink.
- Un module Powerlink BR (**X20 IF 1082-2**).
- Trois supports de présentation détaillant la mise en place d'une communication Powerlink.

Après plusieurs essais, nous nous sommes rendus compte que les entrées / sorties n'étaient pas exploitables directement avec ce matériel. De plus, la procédure détaillée était destinée pour une communication entre un ordinateur et un module d'entrées / sorties déportés. Or le logiciel **FieldBus Designer** ne permet pas de créer de fichier de configuration ni pour l'automate ni pour le module Powerlink fourni. La décision a été prise de changer de matériel et d'utiliser un "Bus Controller Powerlink" BR (**X20 BC 0083**).

Nous avons relié ce module aux éléments suivant :

- L'alimentation 24V Courant Continu (X20 PS 9400).
- Un module de 6 entrées (X20 DI 6371).
- Un module de 6 sorties (X20 DO 6322).

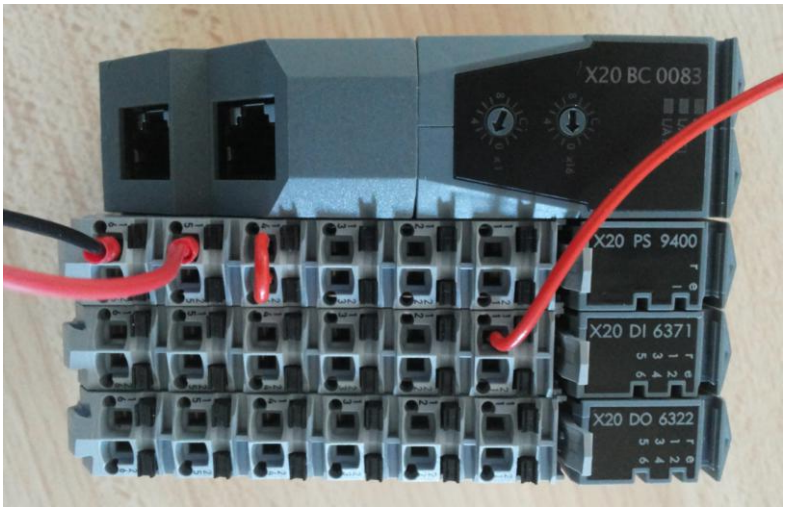


Image 11 : module X20-BC0083 avec ses entrées / sorties

Grâce à ce module d'entrées / sorties déportés, nous avons pu suivre plus facilement la procédure et établir une communication Ethernet Powerlink. Le câblage de *l'image 11* est celui utilisé tout au long du projet. La partie droite concerne l'alimentation du module, la gauche le branchement d'une entrée choisie arbitrairement pour nos tests.

2) Etude du programme console demo_main.c

Une fois toute la configuration établie, nous pouvons exploiter Powerlink en créant nos propres programmes de communication avec l'automate. Pour cela, il convient dans un premier temps de savoir sur quelles fonctions et sur quels paramètres agir pour lire les entrées et modifier les sorties de l'automate.

Comme précisé dans le *chapitre I.3*, la fonction qui nous intéresse se nomme *AppCbSync(void)*. Au téléchargement d'openPowerlink, cette fonction contient un petit programme démo dont le but est de faire clignoter les sorties automate selon l'état des entrées.

Après analyse, nous remarquons que la première fonction appelée permet de récupérer ces valeurs d'entrées et sorties :

```
EplRet = EplApiProcessImageExchange(&AppProcessImageCopyJob_g);
if (EplRet != kEplSuccessful)
{
    return EplRet;
}
```

Image 12 : récupération des entrées / sorties de l'automate

A la suite de cette fonction, nous voyons que le programme récupère **l'état des entrées** à l'aide de la fonction *AppProcessImageOut_g*, puis les copie dans un tableau local :

```
nodeVar_g[0].m_uiInput = AppProcessImageOut_g.CN1_M00_Digital_Input_8_Bit_Byte_1;
nodeVar_g[1].m_uiInput = AppProcessImageOut_g.CN32_M00_Digital_Input_8_Bit_Byte_1;
nodeVar_g[2].m_uiInput = AppProcessImageOut_g.CN110_M00_Digital_Input_8_Bit_Byte_1;
```

Image 13 : copie des valeurs des entrées dans un tableau

A la fin de la fonction, le programme agit **sur les sorties** à l'aide de *AppProcessImageIn_g* en copiant une valeur booléenne sur celles-ci en fonction des opérations relatives à la démo :

```
AppProcessImageIn_g.CN1_M00_Digital_Ouput_8_Bit_Byte_1 = nodeVar_g[0].m_uiLeds;
AppProcessImageIn_g.CN32_M00_Digital_Ouput_8_Bit_Byte_1 = nodeVar_g[1].m_uiLeds;
AppProcessImageIn_g.CN110_M00_Digital_Ouput_8_Bit_Byte_1 = nodeVar_g[2].m_uiLeds;
```

Image 14 : mise à 1 ou 0 des sorties de l'automate

Il est donc simple de changer l'état des entrées et sorties de l'automate une fois ces deux fonctions connues. Dorénavant, il convient de savoir comment utiliser *nos propres* entrées/sorties, celles configurées précédemment. En effet, la démo utilise des entrées (ex : *CN1_M00_Digital_Input_8_Bit_Byte_1*) fictives qui ne nous intéressent pas.

Cette information est contenue dans le fichier **xap.h**. Pour rappel, openConfigurator, une fois le fichier XDC reçu, génère deux fichiers : **mnobd.cdc** et **xap.h**. Le fichier **xap.h** contient toutes les entrées et sorties de l'automate, le programme récupère ces informations pour pouvoir les utiliser ensuite.

Ce fichier **xap.h** est décomposé en deux parties : **PI_OUT** et **PI_IN**, sous forme de structure C. Dans notre cas, voici l'exemple des trois premières entrées :

```
unsigned CN1_M01_X20DI6371_DigitalInput01:1;
unsigned CN1_M01_X20DI6371_DigitalInput02:1;
unsigned CN1_M01_X20DI6371_DigitalInput03:1;
```

Image 15 : trois entrées de notre module X20 DI 6371

Dans cette syntaxe, on reconnait le **type de nœud** (CN), son **numéro** (1), le **placement du module** dans le rack (M01), sa **référence** (X20 DI 6371) et enfin **l'accès à son entrée** (DigitalInput01). Connaissant cette syntaxe, il devient aisé d'utiliser les variables d'entrées dans le programme, exemple :

```
bool value_input_1;
value_input_1 = CN1_M01_X20DI6371_DigitalInput01;
```

Image 16 : récupération de la valeur de l'entrée 1

Il en est de même pour les sorties :

```
// Mise à 0 de la sortie 01 du module X20DO6322
AppProcessImageIn_g.CN1_M02_X20DO6322_DigitalOutput01 = 0;
// Mise à 1 de la sortie 02 du module X20DO6322
AppProcessImageIn_g.CN1_M02_X20DO6322_DigitalOutput02 = 1;
```

Image 17 : mise à 1 ou 0 des différentes sorties

A noter qu'il est également possible de lire l'état des sorties grâce à ces variables :

```
unsigned CN1_M02_X20DO6322_StatusDigitalOutput01:1;
unsigned CN1_M02_X20DO6322_StatusDigitalOutput02:1;
unsigned CN1_M02_X20DO6322_StatusDigitalOutput03:1;
```

Image 18 : lecture de l'état des sorties

Maintenant que nous savons comment utiliser les entrées et sorties de l'automate en fonction des fichiers générés, nous pouvons écrire nos propres applications.

3) Programme C

Une fois la communication établie, nous avons pu nous consacrer à la programmation d'un petit soft dans le but de mettre en avant les capacités de Powerlink. L'objectif a été de mesurer la vitesse des trames sur le réseau, plus précisément, démontrer que le **temps de cycle** est toujours fixé sur la valeur configurée (**10 ms**) peu importe le nombre de données transportées.

Par souci d'ergonomie et de présentation, l'un des premiers essais a été de créer un projet Qt en fenêtre en partant de la base Qt d'openPowerlink. Malheureusement, le programme ayant un nombre de dépendance élevé et une arborescence très précise à respecter (normalement généré par un Makefile), le résultat n'a pas été concluant. Il a donc été décidé de partir du programme console, plus léger, en utilisant les Makefile habituels.

Comme précisé à la fin du *chapitre I.3*, c'est la fonction *AppCbSync(void)* du **demo_main.c** en console qui nous intéresse. C'est celle-ci qui est appelée à chaque temps de cycle. La démo de base évoquée au *chapitre II.2* n'étant pas très intéressante, elle peut être supprimée de la fonction.

a) Calcul du temps de cycle

Maintenant que nous savons sur quelle fonction agir, il suffit d'y inscrire un algorithme capable de calculer le temps successif entre deux appels de fonction. Celui-ci est assez simple, il calcule le temps entre deux appels grâce à l'heure courante, obtenue de manière très précise (en milliseconde), puis l'inscrit dans un fichier texte :

```
/**** CAPTURE DU TEMPS DE CYCLE ****/

gettimeofday(&tv, NULL);
time_in_mill = (tv.tv_sec) * 1000 + (tv.tv_usec) / 1000 ;
// Récupération de l'heure courante en milliseconde

tps_cycle = time_in_mill - time_in_mill_prev;
// On soustrait l'heure courante avec celle récupérée à l'appel précédent

fprintf(fichier, "%f\n", tps_cycle);
// On inscrit le résultat dans un fichier texte

time_in_mill_prev = time_in_mill;
// L'heure courante devient la "valeur précédente"

last_cycle++;

// Petite condition pour une plus belle présentation du fichier texte
if(last_cycle == 10)
{
        fputs("***** END 10 LAST CYCLE TIME *****\n", fichier);
        fseek(fichier, 0, SEEK_SET);
        fputs("***** BEGIN 10 LAST CYCLE TIME (ms) *****\n", fichier);
        last_cycle = 0;
}

/**** CAPTURE DU TEMPS DE CYCLE ****/
```

Image 19 : code du calcul du temps de cycle

1. Récupération l'heure courante en milliseconde.
2. Soustraction de cette valeur avec celle récupérée lors du précédent appel de la fonction.
3. Inscription du résultat de la soustraction dans un fichier texte.

Le fichier texte qui en découle contiendra donc le temps entre les 10 derniers appels de la fonction *AppCbSync(void)*, dont voici un exemple :

```
***** BEGIN 10 LAST CYCLE TIME (ms) *****
11.000000
9.000000
10.000000
9.000000
11.000000
10.000000
10.000000
11.000000
9.000000
13.000000
***** END 10 LAST CYCLE TIME *****
```

Image 20 : le fichier texte avec les temps de cycle

Comme nous pouvons le constater, la valeur est toujours très proche des 10 millisecondes attendues. Il est tout à fait probable que la marge d'erreur (à + ou - 1 milliseconde) vienne du code en lui-même (récupération de l'heure courante, inscription dans un fichier texte…).

b) Compteur de palettes

En plus du calcul du temps de cycle, un second programme a été créé pour tester les performances du réseau un peu plus concrètement. Son but est d'afficher les passages à « **1** » d'une entrée choisie arbitrairement et de les compter successivement.

Ce mini-programme pourrait être une simulation d'un compteur de palettes d'une usine dont les passages devant un capteur sont rapides. L'intérêt de Powerlink dans ce cas-là est de pouvoir tester son aptitude à "garder le compte" malgré la vitesse de passage. Voici le code en question :

```
/**** PROGRAMME UTILISATEUR CPT PALETTE ENTREE 1 ****/

state_in_cur = AppProcessImageOut_g.CN1_M01_X20DI6371_DigitalInput01;
// On récupère l'état de l'entrée 1 dans la variable state_in_cur

// Si cette valeur vaut 1 et qu'elle est différente de l'état précédent..
if (state_in_cur == 1 && state_in_cur != state_in_prev)
{
        cpt_pal++;
        // On incrémente un compteur
        printf("Cpt palette = %d\n", cpt_pal);
        // On affiche la valeur de celui-ci dans la console
}

state_in_prev = state_in_cur;
// La valeur courante de l'entrée devient la précédente

/**** FIN PROGRAMME UTILISATEUR CPT PALETTE ENTREE 1 ****/
```

Image 21 : code du programme compteur de palettes

1. Récupération de l'état de l'entrée 1 du module d'entrées de notre automate
2. Comparaison de cette valeur avec son état précédent. Si cette entrée est passée à « **1** », on incrémente un compteur et on l'affiche sur la console.

Une fois le programme lancé, la console affiche correctement la valeur du compteur à chaque passage à 1 de l'entrée.

```
2014/04/16 10:33:23 - AppCbEvent(Node=0x1, ConfReset)
2014/04/16 10:33:23 - AppCbEvent(Node=0x1, Found)
2014/04/16 10:33:26 - AppCbEvent(Node=0x1, NmtState=NmtCsReadyToOperate)
2014/04/16 10:33:26 - AppCbEvent(Node=0x1, NmtState=NmtCsOperational)
Cpt palette = 1
Cpt palette = 2
Cpt palette = 3
Cpt palette = 4
Cpt palette = 5
Cpt palette = 6
Cpt palette = 7
Cpt palette = 8
Cpt palette = 9
Cpt palette = 10
Cpt palette = 11
Cpt palette = 12
```

Image 22 : affichage de la console : compteur de palettes

Des tests rapides nous ont permis de constater que Powerlink détectait n'importe quel passage même si nous passions l'entrée successivement de 0 à 1 très rapidement. Cependant, l'expérimentation étant faite « à la main », il est difficile de tirer des conclusions définitives.

4) Parallèle avec le protocole Modbus

Pour mettre en avant les performances de l'Ethernet Powerlink, une deuxième expérimentation consistait à utiliser le protocole Modbus dans le but de comparer les protocoles. Pour cela, nous avons communiqué avec un second automate en utilisant Modbus et écrit un petit programme qui lit les valeurs d'une entrée toutes les **10 ms**.

Pour écrire ce programme, nous nous sommes basés sur l'application **Modbus client** du TP réseau industriel EI4. Grâce à lui, il est possible de se connecter à un automate mais surtout, de lancer une lecture périodique des variables mémoires internes de ce dernier. Nous avons donc réglé ce temps d'appel à **10 ms** pour rester dans les temps Powerlink, puis écrit notre propre code dans la fonction en question, appelée périodiquement.

L'application codée ressemble très fortement à celui utilisé dans le **chapitre II.3**, excepté que le « faux temps de cycle » est affiché directement sur la console, et non plus dans un fichier texte :

```
/***** AFFICHAGE DU TEMPS DE CYCLE TOUTES LES 10 MILLISECONDES ****/

gettimeofday(&tv, NULL);
time_in_mill = (tv.tv_sec) * 1000 + (tv.tv_usec) / 1000 ;

tps_cycle = time_in_mill - time_in_mill_prev;

qDebug() << tps_cycle; // Affichage du pseudo "temps de cycle" entre deux executions de la fonction

time_in_mill_prev = time_in_mill;
```

Image 23 : code du calcul du « temps de cycle » sur Modbus

Voici les résultats une fois le programme lancé :

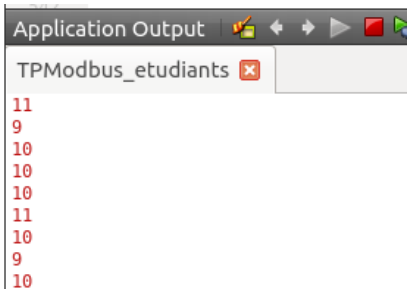


Image 24 : résultats obtenus lors du lancement du programme

Nous remarquons que Modbus arrive tout à fait à lire notre variable dans les temps. En revanche, lorsque nous lançons une écriture, le programme ralentit légèrement.

Image 25 : lancement du programme avec une écriture

Cependant, avec le recul, ce type d'application ne prouve pas grand chose dans la mesure où Modbus n'utilise pas de temps de cycle à proprement parlé. Ce qu'il aurait fallu mettre en évidence, ce sont surtout les problèmes de collisions. Il faudrait malheureusement plusieurs équipements actifs pour effectuer ce genre d'expérience.

Le code utilisé a tout de même le mérite de montrer que Modbus n'est pas déterministe car il n'est pas calé sur un temps de cycle défini. Ici, l'appel de la fonction périodique est faite d'un point de vu code mais n'est pas lié au protocole Modbus lui-même.

5) Perspectives d'évolution

Lors de ce projet, des tests de rapidité ont été effectués sur les protocoles Powerlink et Modbus. Les temps de cycle sont équivalents, autour de 10ms. On remarque toutefois que le protocole Modbus a des pics à 20ms alors que Powerlink ne dépasse pas les 15ms.

Nous pouvons en conclure que le protocole Powerlink est plus constant que le protocole Modbus. C'est assez logique dans la mesure où le protocole Powerlink fonctionne grâce à des temps de parole alors que le protocole Modbus envoie des requêtes et attend la réponse des équipements.

Le temps de cycle utilisé pour notre projet est plutôt important. Ceci est en partie expliqué par la carte Ethernet utilisée, qui est une carte standard donc limitée à 10ms. Une amélioration possible serait d'utiliser une carte Ethernet plus rapide ou de la configurer en temps réel.

Nous pouvons penser que sur un réseau de terrain plus conséquent et actif, le protocole Modbus serait beaucoup moins rapide que Powerlink. En effet, le protocole Modbus utilise Ethernet donc perd du temps à gérer les collisions alors que Powerlink les évite grâce à des temps de parole définis pour chacun. On pourrait utiliser plusieurs machines pour tester l'impact des collisions sur le temps de cycle.

Conclusion

L'objectif du projet était la réalisation d'une maquette de TP. Bien que celui-ci n'ait pas été réalisé, la première étape a été faite : une connexion Powerlink a été établie et des programmes de démonstration réalisés.

Le premier objectif du TP étant la réalisation d'une communication Powerlink, le guide openPowerlink pourrait être transformé en sujet de TP en y ajoutant des questions. Par la suite, il serait possible d'utiliser les programmes réalisés comme correction de TP en construisant les questions autour de ceux-ci.

Lors de ce projet, nous nous sommes concentrés sur la configuration de la connexion Powerlink et la création des fichiers de configurations (mnobd.cdc et xap.h). Une autre possibilité aurait été de demander ces fichiers a l'entreprise B&R, ce qui nous aurait permis de passer plus de temps élaborer un sujet de TP et de le tester. Nous n'avons pas retenu cette solution car il nous paraissait important que la configuration de la connexion soit intégrée dans le TP.

Ce projet nous a permis d'utiliser nos connaissances en commande Unix, programmation C et réseau de terrain et de les solidifier. Nous avons pu acquérir de nouvelles compétences sur le fonctionnement d'Ethernet Powerlink et sa mise en oeuvre, mais aussi les procédures de génération via Makefile ainsi que le travail de groupe.

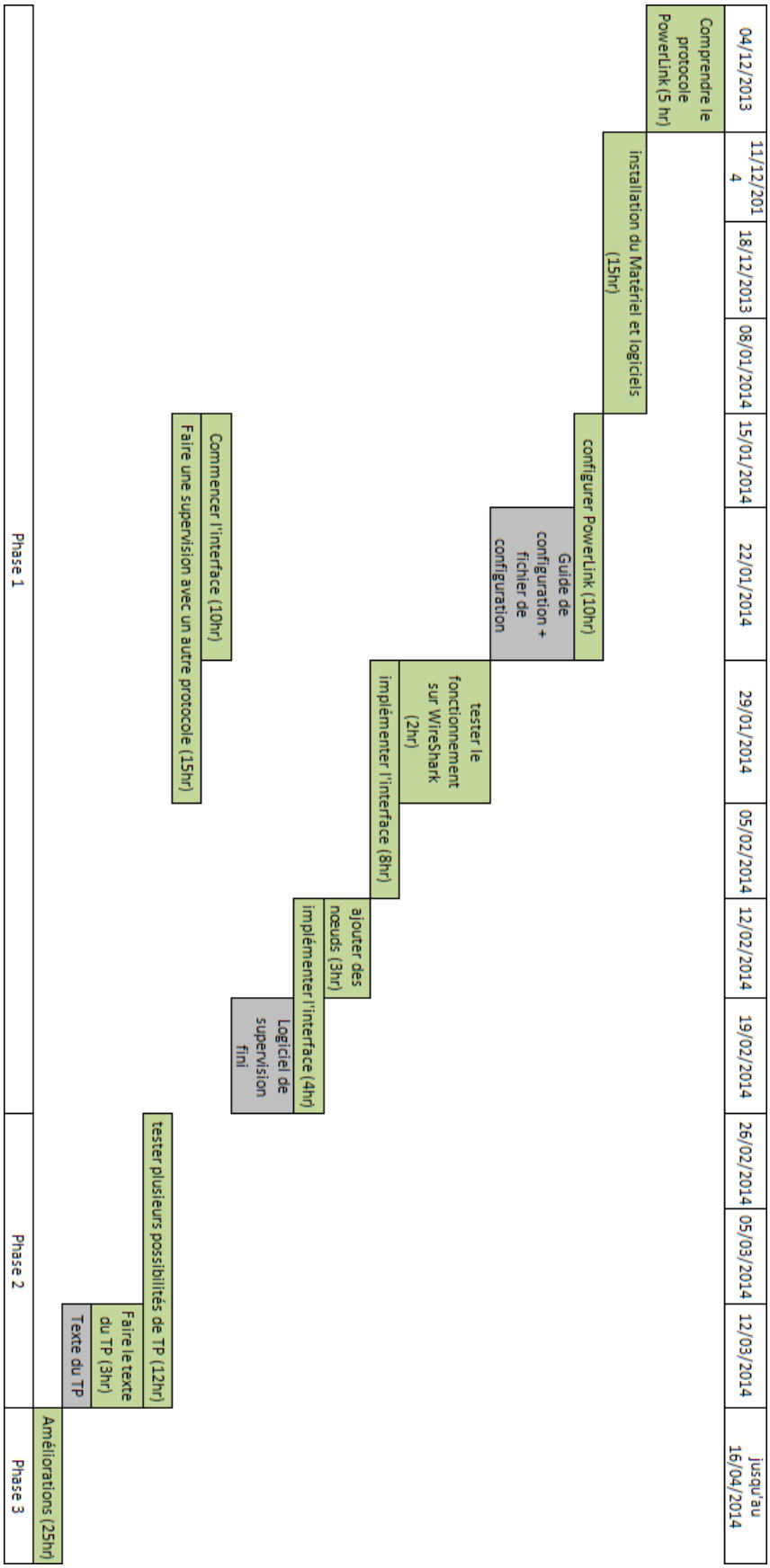
Bibliographie

IXXAT, fournisseur de produits dans les solutions systèmes embarqués. [En ligne]
http://www.ixxat.fr/powerlink_technologie_fr.html

Kalycito, participation à la création d'openConfigurator. [En ligne]
http://www.kalycito.com/cms/

Ethernet Powerlink, site officiel du protocole Powerlink. [En ligne]
http://www.ethernet-powerlink.org/

Annexes



<u>Annexe 1 :</u> diagramme initial de Gantt

Maquette pour un TP de réseau de terrain Ethernet Industriel temps réel

Projet réalisé par : **CUINIER Léna, DARROU Damien, LAAOUINA Marouane**
Projet encadré par : **M. LAHAYE Sébastien, M. LAGRANGE Sébastien**

L'objectif du projet était la création d'une maquette de TP démontrant, aux étudiants, l'utilité de Powerlink. Les réseaux de terrain sont utilisés pour connecter ensemble des automates, des entrées-sorties déportés et des ordinateurs. Ils sont utilisés pour créer des processus industriel et les superviser. Powerlink est connu pour être l'un des plus rapides.

Nous avions à disposition des ordinateurs, un automate ainsi que des documents contenant une procédure permettant d'établir une connexion Powerlink. La première étape a été de suivre cette procédure. Après un mois de documentation et de tests, nous nous sommes aperçus que le logiciel utilisé ne correspondait pas à l'automate.

La deuxième étape a été l'achat d'un module d'entrées-sorties déportés et la création de la communication avec un ordinateur. Ensuite, le code du logiciel a été modifié pour mettre en avant la rapidité de la communication. Dans un même temps, un guide a été écrit pour expliquer la façon dont la connexion a été établie. Finalement, un programme similaire qui utilise Modbus (un autre réseau de terrain) a été créé pour faire une comparaison avec Powerlink.

Mots-Clés : *Powerlink, entrées-sorties, connexion, code, guide, comparaison, constante, vitesse, Ethernet.*

The goal of the project is to create an experiment that demonstrates uses of Powerlink to students. Fields networks are used to connect together Programmable Logic Controller (PLC), input/output device or computer. They are used to design industrial processes and to supervise them. Powerlink is known for being one of the quickest field networks.

Computers, a PLC and documents containing a procedure to establish a Powerlink connection were available. The first step was to follow this procedure. After a month of documentation and experiments, we realized that the software we had did not correspond to the PLC.

Second step was to buy an input/output device and make it communicate with a computer. Then, the software code was modified to show the speed of the exchange. At the same time a specific guide was written to explain how the connection had been established. Finally, a similar program was built using Modbus (another field network) to compare with Powerlink.

Keywords: *Powerlink, input/output, connection, code, guide, compare, constant, speed, Ethernet.*