DIFFUSION : Confidentielle [ ] Restreinte [ ] Contrôlée [x] Non Contrôlée [ ]

Document d'architecture d'un système de motorisation électrique
V0 IE07.1

DIFFUSION

Libre

2016 06 02 QUESTIONS JMR :
- page 16 : la machine d 'état proposé, est ce pour le reéseau ou est ce l'état d'un esclave ?
- page 20, c'est quoi le mode repli ?

IND	DATE	ETABLI	VERIFIE	APPROUVE	MODIFICATIONS
IE07.2	24/02/17	J.M. Routoure	N/A	N/A	Corrections sur la variable de pédalage envoyée sur le BIONET
IE07.1	02/06/16	J.M. Routoure	N/A	N/A	Corrections de coquilles/fautes de frappe
IE07	29/10/2015	F Trocque	N/A	N/A	Modif Variable Variable DC22
IE06	30/04/15	F Trocque	N/A	N/A	Ajout variable pédalage DC22
IE05	23/04/15	F Trocque	N/A	N/A	Change Hardware & Parité (sans) description pour BIO-Net Supprime MOS cana P Ajoute exigence DC11 (RPM vélo) Modifie variable vitesse par RPM (Rotation par minute)
IE04	13/11/14	Trocque	N/A	N/A	Change variable référence (seulement 128 variable possible B7=R/W) Ajout trame Gestion réseau
IE03	28/10/14	Trocque	N/A	N/A	Modification BIO-Net, ajout exigences, Capteur courant passe dans DC11, ajout capteurs physiques

	DATE	NOM	VISA
ETABLI	F Trocque		
VERIFIE	N/A		
APPROUVE	N/A		

Table des matières

1. Introduction

1.1. Objectifs

Le projet a pour objectif, la réalisation d'un ensemble de composant électrique, permettant l'électrification d'un véhicule de petite puissance, de type vélo électrique

1.2. Hypothèses/Contraintes

Les constituants réalisés doivent constitués un ensemble de composants, sur étagère, pouvant s'interconnecter et permettant ainsi la réalisation de systèmes électriques simples à sophistiqués
L'ensemble devra offrir la possibilité de s'adapter à différent types de moteur
Une fonction de connexion à un appareil de type Smartphone devra être disponible

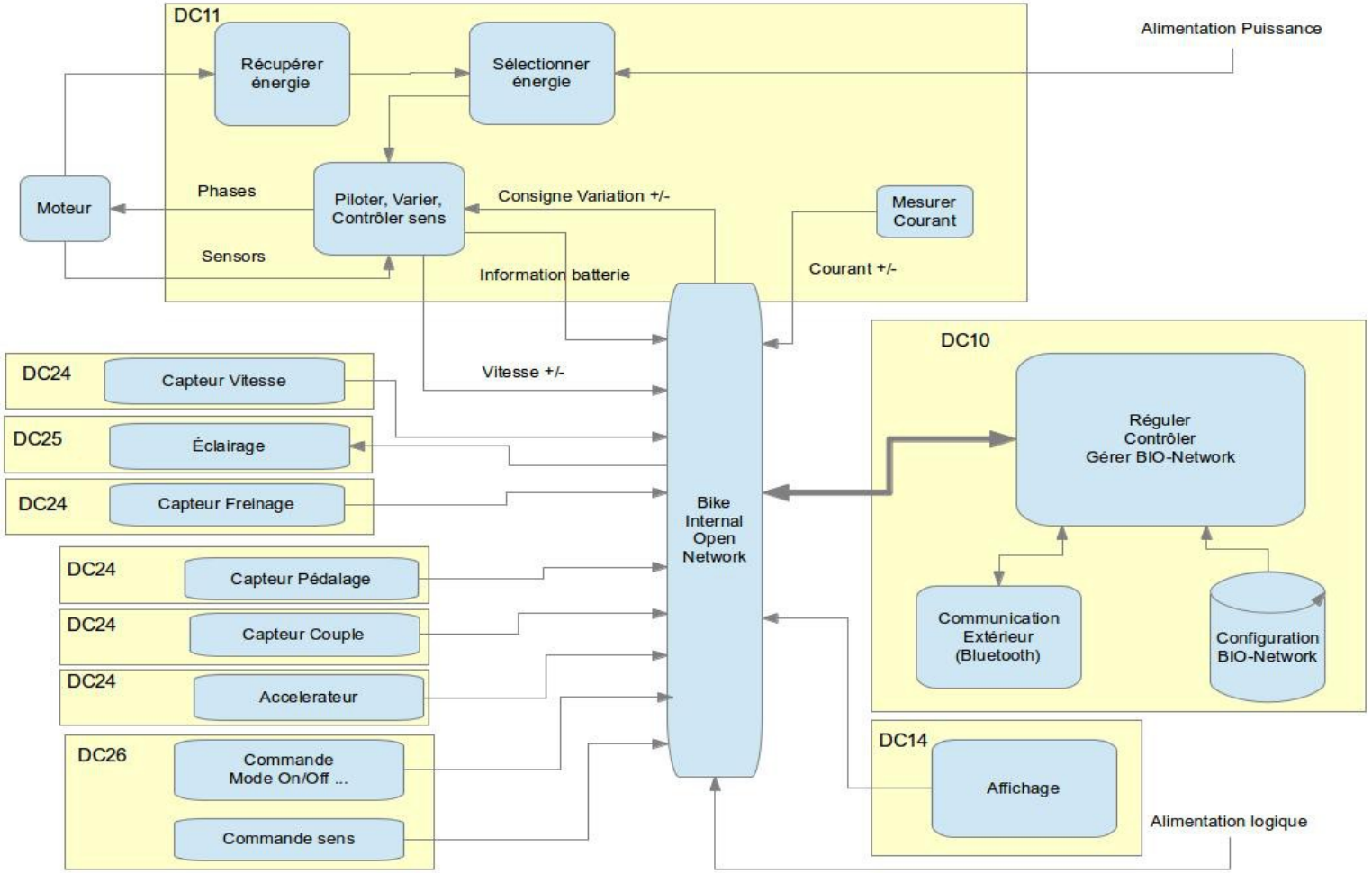
1.3. Documents de références

1.4. Abréviations

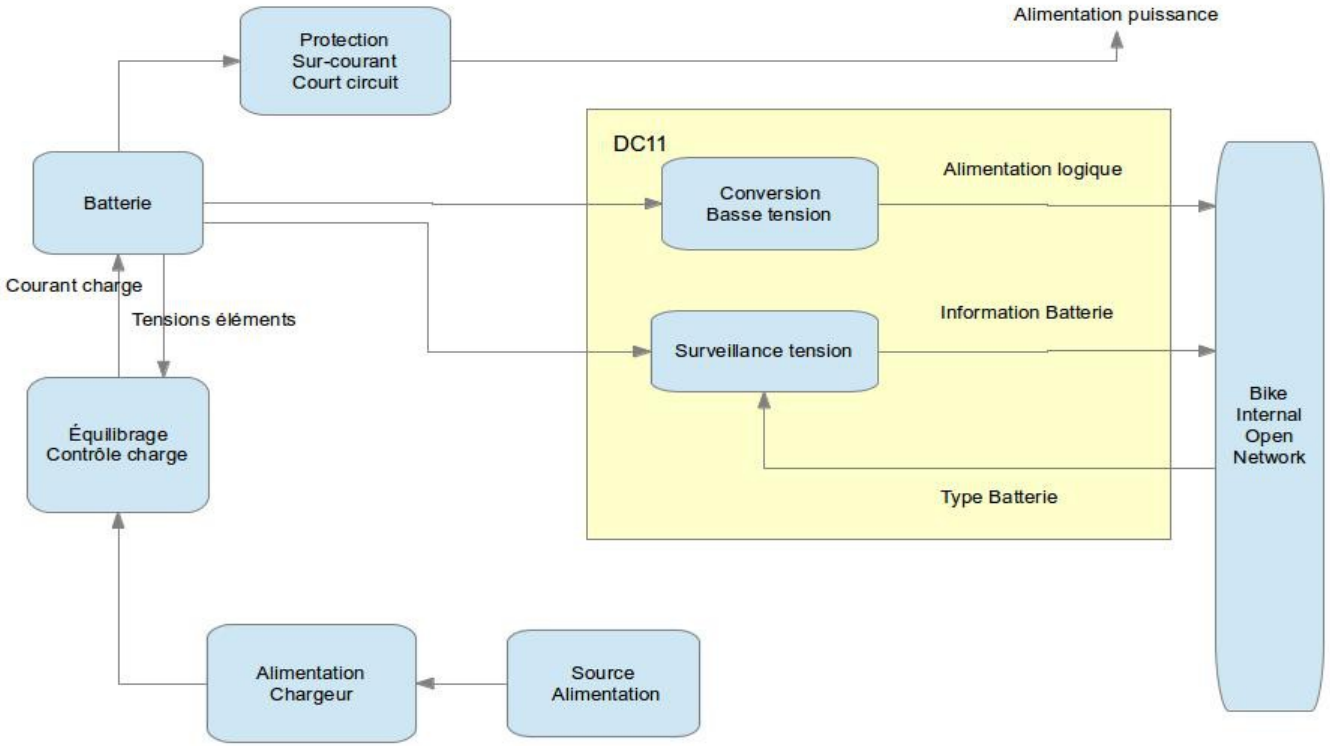
IHM Interface Homme Machine
PWM Pulse Width Modulation (modulation de largeur d'impulsion)
RUF Réservé Usage Future
RPM Rotation par minute

2. Architecture générale

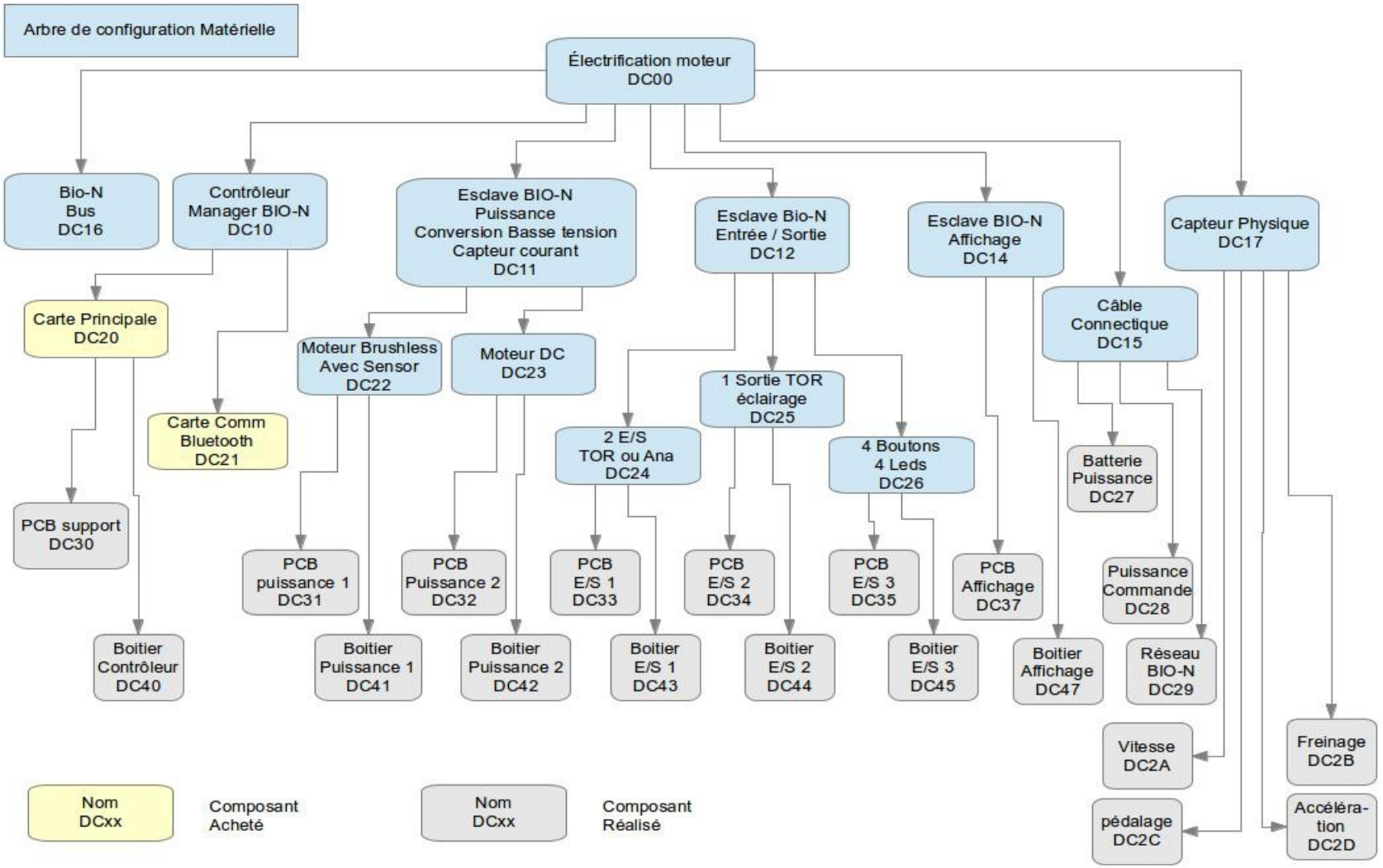
2.1. Partie contrôle, variation



2.2. Partie Alimentation, Batterie



2.3. Arbre de configuration matériel



3. Architecture système minimale

Architecture minimale du système de pilotage d'un vélo électrique,est composé au minimum

- DC10 Contrôleur
- DC22 ou EC23 esclave variateur
- DC24 fonction capteur de pédalage
- DC24 fonction capteur de vitesse

4. DC10 Contrôleur Manager

5. DC11 Puissance & Conversion basse-tension

5.1. Exigences

REQ-11-001 Assurer la variation du courant du moteur (variable Consigne)
REQ-11-002 Consommer la variable Consigne
REQ-11-003 Mesurer le courant du moteur
REQ-11-004 Produire la variable Courant
REQ-11-005 Mesurer la valeur de la tension Batterie (puissance)
REQ-11-006 Produire la variable Information batterie
REQ-11-007 Consommer Seuil batterie
REQ-11-008 Stopper le moteur si Vbat < seuil
REQ-11-009 être compatible avec BIO-Net
REQ-11-010 Convertir Vbatt (puissance) en +5V pour tout le bus BIO-Net et ses esclaves
REQ-11-011 Permettre la connexion en amont du convertisseur basse tension d'un interrupteur (ON-OFF système)
REQ-11-012 Consommer moins de 1mA sur Vbatt lorsque le système est OFF
REQ-11-013 visualiser l'état du réseau (LED verte INIT flash 0,1s/1s, RUN on, STOP clignontement 1s/1s, ERREUR flash 0,1s/0,3s)
REQ-11-014 Mesurer la vitesse du vélo en Km/h lorsque le moteur tourne

5.2. Processeurs utilisés

Processeur utilisé 16F1509 Microchip 1,1€ / 25 chez Conrad

L'UAR de ce processeur est identique à celui du 16F1822
Avantage : une seule stack dialogue à développer

DC 11 est décliner en au moins 2 sous ensembles DC22 Moteur brushless et DC23 Moteur DC

5.3. DC22 Moteur Brushless

5.3.1. Exigences

REQ-22-00 Piloter les 3 phases du moteur de manière libre (freewheeling) pour DC22

5.3.2. Description

Ce module de puissance pilote des moteurs axiaux équivalents à celui utilisé sur le Kit acheté
Ces moteurs sont des moteurs synchrones triphasés.
Il contiennent 3 capteurs à effet Hall permettant d'analyser la position du rotor.

5.3.3. Pilotage des moteurs

Le pilotage des ces moteurs nécessite, 3 entrées pour les capteurs, 6 sorties afin de piloter les 3 ponts en H connectés à chacune des phases du moteur

Afin de simplifier le design du schéma électronique, on utilisera des composants réalisant les fonctions essentielles. Ils assureront aussi une parfaite gestion des fronts de montée et descentes, des 2 Mos, limitant ainsi au minimum les pertes en régime transitoire

- High Side driver → IRS21850 (1,08€ HT /100 farnell) *3 dans le design
- Low side driver → IR4426 (1,44€ HT /100 farnell 1,62€TTC /25Conrad) dual, donc *2 dans le design

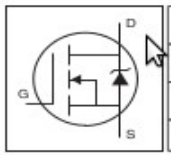
Ces 2 drivers sont en boîtier SO8

MosFet canal N
AUIRF1010 (0,87 HT /100 farnell) *3 dans le design

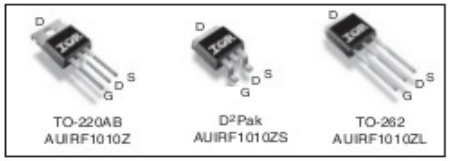
PD - 97458A

AUIRF1010Z
AUIRF1010ZS
AUIRF1010ZL

HEXFET® Power MOSFET



$V_{(BR)DSS}$	55V
$R_{DS(on)}$ max.	7.5mΩ
I_D (Silicon Limited)	94A
I_D (Package Limited)	75A



Puissance dissipée par les MOSFet.

Sur nos 3 ponts en H, uniquement un pont est actif à chaque instant, la puissance dans chacun des ponts, correspond au 1/3 de la puissance équivalente dissipée continue.
Pour une batterie de 36V et 250W maximum, le courant circulant dans le moteur est d'environ 7A

Pour les MOS High side & Lowside :
La puissance dissipée par effet Joules dans le transistor à l'état passant est de P= 7*7* 0,0075 /3 soit 0,13 W. Un radiateur n'est pas nécessaire.

5.3.4. Contrôle et mesure du courant

Afin de réduire le nombre de composant dans le système, on intègre une mesure de courant à DC22
On utilisera le composant suivant :
ACS712ELCTR-20A-T (2,68€ /100 Farnell)
C'est un capteur à effet Hall +/- 20A boitier SO8

5.4. DC23 Moteur DC

Développé ultérieurement

Il pourra être totalement identique à DC22 à l'exception du nombre de MOS implémentés
On peut imaginer 2 versions

- 1 avec un seul MOS Low side → moteur DC un sens de rotation
- 1 avec 2 ponts en H → moteur DC deux sens de rotation

6. DC12 Esclave Entrées / Sorties

6.1. Exigences

REQ-12-001 être compatible avec BIO-Net
REQ-12-002 Alimentation prise sur le bus

REQ-12-003 Produire la variable IHM accélération
REQ-12-004 Produire la variable Pédalage
REQ-12-004 Produire la variable freinage
REQ-12-005 Produire la variable vitesse
REQ-12-006 se connecter au capteur accélérateur (analogique)
REQ-12-007 se connecter au capteur pédalage
REQ-12-008 se connecter au capteur freinage
REQ-12-009 se connecter au capteur vitesse
REQ-12-010 assurer un moyen de sélection du ou des capteurs connectés
REQ-12-011 visualiser l'état du réseau (LED verte INIT flash 0,1s/1s, RUN on, STOP clignotement 1s/1s, ERREUR flash 0,1s/0,3s)
REQ-12-012 Assurer une connectique étanche avec le capteur

6.2. Processeur utilisé

Processeur utilisé 16F1822 Microchip 0,9€ / 25 chez Conrad

L'uart de ce processeur est identique à celui du 16F1509
Avantage : une seule stack dialogue à développer

On utilisera l'horloge interne du processeur afin de réduire les coût.

Normalement l'oscillateur interne est réglé lors de la fabrication du composant à +/-2 % (de 0° à 60°) . Cette valeur ne posera pas de problème pour la communication série, puisque la tolérance nominale d'une telle liaison des de 5 %
Toutefois sous 0°, le composant peut atteindre +/- 5 %, on arrive alors la limite de fonctionnement
(un vélo peut être utilisé avec des températures négatives)
Il faudra prévoir l'emplacement pour un quartz au cas où.

7. DC14 Esclave affichage

TBD

8. DC15 Connectique

TBD

C'est un point important et coûteux du système.

Si on compte les connecteurs nécessaires à la réalisation d'un esclaves, il faut idéalement

- 1 entée Bus
- 1 sortie Bus
- 1 connecteur E/S

Cette liste de connecteurs implique l'utilisation de connecteurs complémentaires sur les câbles

Soit un total de 6 connecteurs pour un point de connexion. !!!!!

De plus les connecteurs doivent être **étanches**.

Une solution pourrait être de ne pas utiliser de connecteurs, mais d'avoir des esclaves avec des emplacement pour souder directement les cables sur le PCB.
Une fois le montage valider, on pourrait couler une résine dans le boitier de l'esclave, afin de rendre l'ensemble étanche et indestructible.

- Avantages
  Coût
- Inconvénient
  Indémontable
  Pas de maintenance possible sans couper et donc réduire la taille des câbles

9. DC16 Bio-N Bus

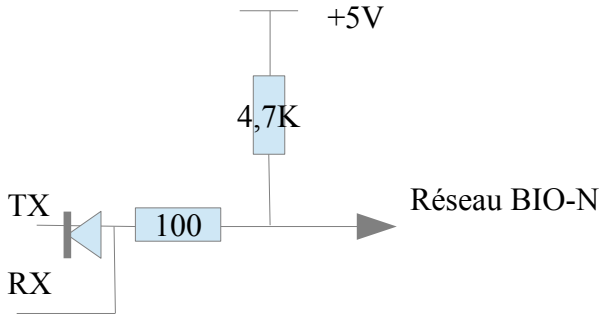
9.1. Description générale

Le bus est basé sur une liaison série de type asynchrone utilisant un protocole type maître esclave
Le management global du réseau sera confié à DC10 (Contrôleur manager)

La couche physique sera réduite à sont minimum et ne sera pas sans rappeler une boucle de courant

9.2. Couche physique

Schéma électrique d'un point de connexion.



L'idée est bien évidemment de réduire le coût de connexion au minimum.
Le micro contrôleur est directement connecté au bus. Une résistance de 100 Ω vient protéger le port. Un condensateur de faible valeur pourra être ajouté aux bornes RX et TX du contrôleur afin de réaliser un léger filtrage (attention à la vitesse de transmission)

La communication est de type asynchrone 19200 bds, 8 bits, sans Parité, 1 stop bit

→ La parité n'est pas calculée par hardware sur le processeur

TBC : La réalisation de ce type de schéma, n'est possible que si le port de transmission est paramétrable en mode collecteur ouvert

TBC : Vérifier la vitesse maximale atteignable sur le bus pour une longueur de 2m. Il faut atteindre 19200bds afin d'avoir des temps de cycles court pour le rafraîchissement des données de procédé
S'il nest pas possible d'atteindre cette vitesse, il faudra passer probablement sur un bus RS485. Ce qui n'est pas souhaitable en terme de coût

9.3. Protocole de communication

Le protocole de communication est basé sur un adressage de variable, plutôt que sur un adressage d'esclave.

Ce qui signifie que les trames véhiculées sur le bus, seront des trames de lecture ou d'écriture de variables. Elle n'auront pas connaissance de la position physique de l'esclave qui consommera ou produira cette variable

Les trames seront du type question réponse. Il y aura donc un Maître de réseau.
Les informations seront donc toutes échangées du maître vers l'esclave ou vice versa. Aucune information ne passera directement d'un esclave vers un autre
Ce réseau est un mixte entre un modèle producteur, consommateur et un modèle maître esclave

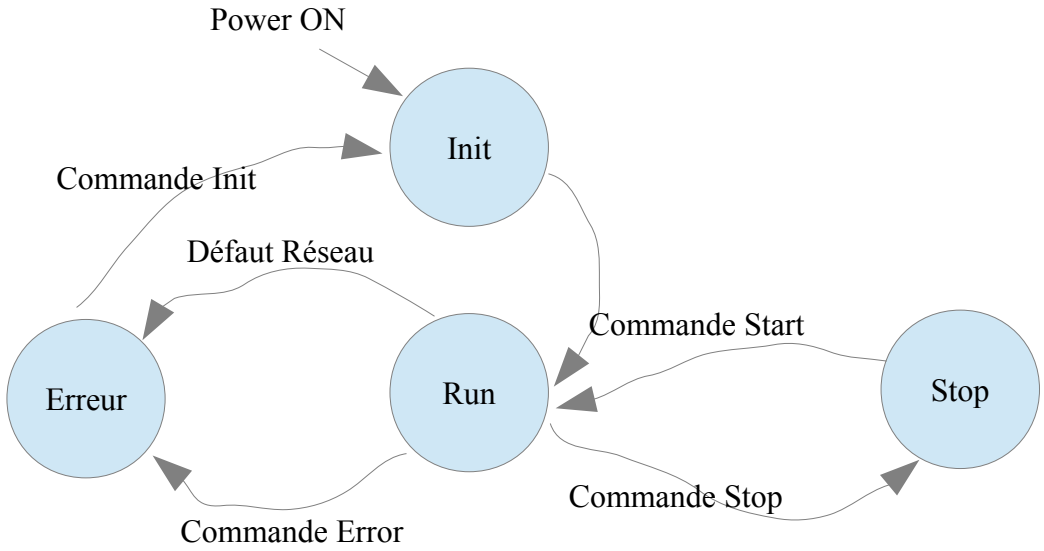
Avantage :

- pas de paramétrage d'adresse sur les esclaves → pas de coût supplémentaire (switch, programmation …..)
- Le manager ne connaît pas la composition physique du réseau. Il peut s'adapter à toute structure réseau contenant ces variables

Inconvénient :

- Une trame par variable. (on peut avoir des variables complexes → structures)

9.3.1. Diagramme d'état du réseau



Le bus possède 4 états : Init, Start,Stop & Erreur
INIT → dans cette phase, le manager de réseau va interroger toutes les variables qu'il gère afin d'identifier la composition du réseau
Si la composition est OK et gérable, le manager passera le réseau en RUN, sinon l'état ERREUR sera demandé
Dans cette phase, la lecture des variables renvoie la taille ainsi que la période souhaitée d'échange en multiple du cycle de base (10ms)

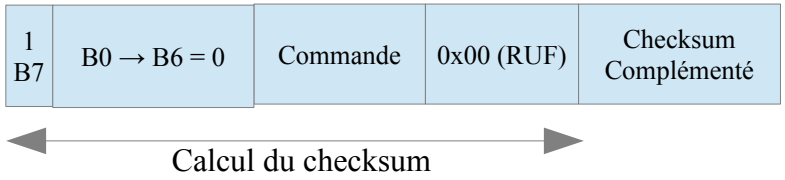
RUN → Dans cette phase, les échanges de variables sont réalisés et chaque esclave gère ses « timeouts » associés à chaque variable.
En cas de défaut TimeOut sur une variable, l'esclave passe dans l'état ERREUR

ERREUR → Dans ce mode l'esclave ne traite plus aucun message. Il scrute uniquement la variable de commande réseau pour éventuellement passer dans l'état INIT

STOP → Dans ce mode l'esclave continue à répondre aux variables, mais ne gère pas les timeout.
Les écritures de variables sont répondues au point de vue réseau, mais les valeurs écrites ne sont pas appliquées

9.3.2. Trame gestion réseau

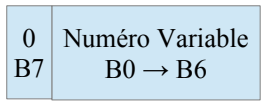
Diffusion (Maître → Esclave)



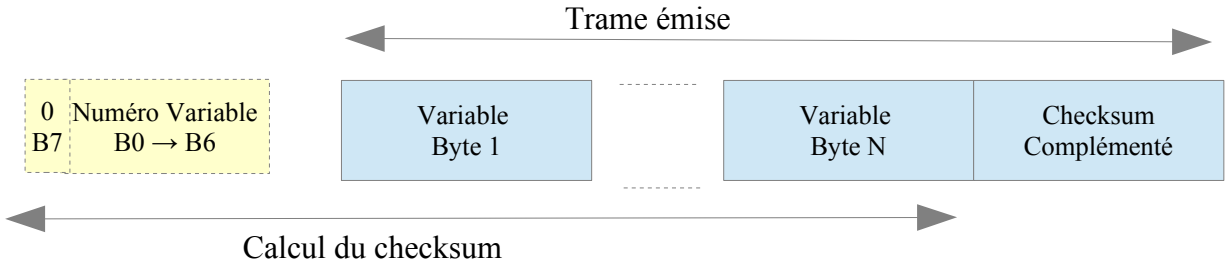
Commande :
-0 Passage en mode INIT
-1 Passage en mode RUN
-2 Passage en mode STOP
-3 Passage en mode ERROR

9.3.3. Lecture de variable dans l'état RUN & STOP

Question (Maître → Esclave)



Réponse (Esclave → Maître)



La réponse de l'esclave n'est envoyée que s'il reconnaît le numéro d'une variable qu'il gère

Champ de contrôle :

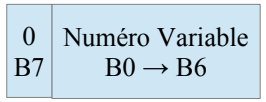
Il est constitué d'un checksum calculé sur le numéro de la variable reçu, ainsi que les octets de données envoyés pas l'esclave.
Le checksum est envoyé complémenté

Il permet de gérer les cas suivants.

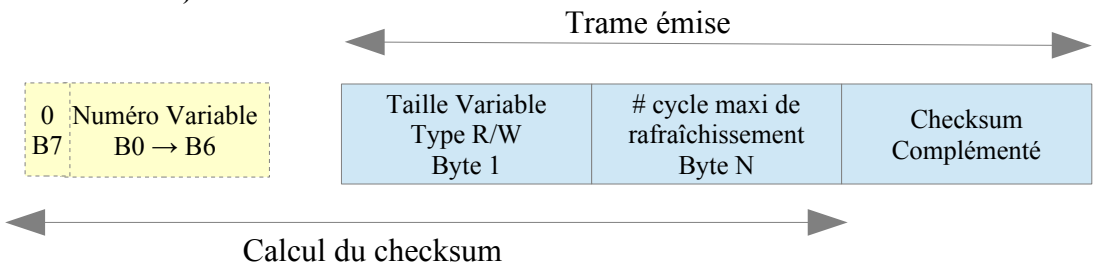
- Erreur lors de la réception du numéro de variable, détectée au niveau du maître
- Erreur lors de la réception par le maître du contenu de la variable

9.3.4. Lecture de variable dans l'état INIT

Question (Maître → Esclave)



Réponse (Esclave → Maître)



La réponse de l'esclave n'est envoyée que s'il reconnaît le numéro d'une variable qu'il gère.

Taille variable est codée sur 3 bits B0 B2 (0 à 6 octets) afin de rester dans le cycle de 10ms
6 octets en écriture + 2 encapsulations + 1 réponse = 9 → 0,5ms par caractère = 4,5ms (toute la bande passante est prise)
Type Variable est codé sur 2 bits B4 & B5
B5=0,B4=1 Lecture
B5=1, B4=0 Écriture

Cycle maxi de rafraîchissement → 1 à 255 soit 0,01s à 2,55s

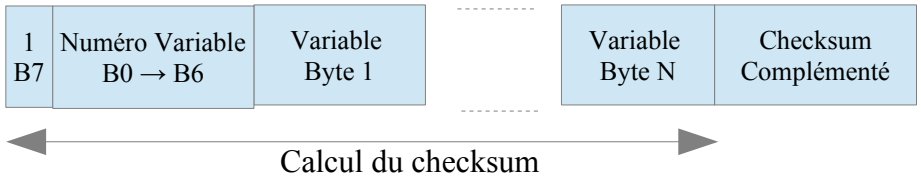
Toute ces informations seront utilisées par le manager afin de créer sa recette de gestion du réseau.

Champ de contrôle :

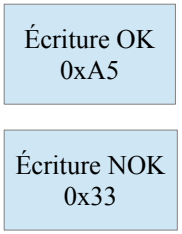
Il est constitué d'un checksum calculé sur le numéro de la variable reçu, ainsi que les octets de données envoyés par l'esclave.
Le checksum est envoyé complémenté.

9.3.5. Écriture de variable dans l'état RUN & STOP

Question (Maître → Esclave)



Réponse (Esclave → Maître)



Champ de contrôle :

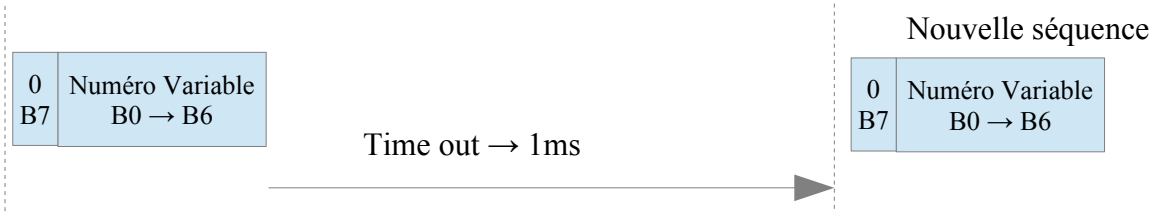
Il est constitué d'un checksum calculé sur le numéro de la variable, ainsi que les octets de données envoyés à l'esclave.
Le checksum est envoyé complémenté

Il permet de gérer les cas suivants.

- Erreur lors de la réception de la trame (au niveau esclave)→ pas de réponse sur le réseau

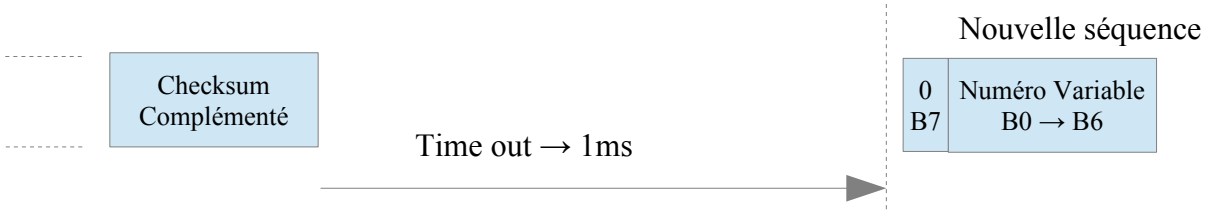
9.3.6. Gestion des erreurs

Non réponse de l'esclave lors de lecture



Après l'émission du numéro de variable, le maître arme un timeout de 1ms.
Si l'esclave n'a pas répondu dans les 1ms, le maître passera à la séquence suivante

Non réponse de l'esclave lors de l'écriture



Après l'émission du checksum, le maître arme un timeout de 1ms.
Si l'esclave n'a pas répondu dans les 1ms, le maître passera à la séquence suivante

Collision

Compte tenu du principe d'échange Maitre Esclave, la collision n'est normalement pas possible, si toutefois elle survenait, alors nécessairement une erreur sur le champ de contrôle serait détectée

Comptage des erreurs et état du réseau

Coté Maître

Chaque variable lue ou écrite dispose d'un compteur d'erreur associé
Lors de la non réception d'une réponse à la lecture ou l'écriture d'une variable, le maître incrémentera le compteur d'erreur spécifique à la variable.

Dés qu'un compteur d'erreur atteint la valeur 10 (TBC), Le maître déclarera le réseau comme HS et stoppera toute communication
L'arrêt de la communication fait passer tout les esclaves en mode replis

Le compteur d'erreur de la variable est décrémenté à chaque réception correcte de la réponse associée à la variable (jusqu'à la valeur 0)

Coté esclave

Chaque esclave armera un timer de 500ms dès qu'une de ces variables est lue ou écrite correctement.
Si le timer arrive à échéance, l'esclave passe en mode replis.

Le mode replis n'a de sens que pour les esclaves pilotant une sortie. L'esclave se placera dans mode assurant la sécurité.
Ex : puissance, mode replis = moteur OFF

9.3.7. timing du réseau

Timing lecture

Un échange en lecture est composé de 3 octets minimum pour une donnée de 1 octets, pour le transport d'un nombre entier, il faudra 4 Octets.
En imaginant un temps de retournement de l'esclave de 0,5ms, on obtient :
2 ms pour 1 octet
2,5 ms pour un entier (2 octets)

Timing écriture

Un échange en écriture est composé de 4 octets minimum pour une donnée de 1 octets, pour le transport d'un nombre entier, il faudra 5 Octets.
En imaginant un temps de retournement de l'esclave de 0,5ms, on obtient :
2,5 ms pour 1 octet
3 ms pour un entier (2 octets)

Cycle de base du réseau

Sur notre réseau 2 données de procédé doivent être échangées rapidement :

- la mesure courant
- La consigne de variation

Toutes 2 sont des variables de type entier (2 octets)

Notre cycle minimum sera donc de : 2,5 + 3 =5,5ms

Si on se fixe comme objectif d'échanger les données de procédés toutes les 10ms, il nous reste 4,5ms de disponible.
Il serait donc possible de décomposer le cycle de base en :

- 2 échanges périodiques (courant, consigne)
- 1 échange apériodique (lecture ou écriture des autres variables)

Sur cette base de calcul, on peut imaginer échanger jusqu'à 100 variables par seconde.

Le temps de cycle de 10 ms est pris simplement sur la base de l'expérience de mon propre vélo électrique dont le contrôleur est cadencé à 10ms
Un temps de cycle de 20 ms ne devrait pas être préjudiciable, nous avons alors encore 50 corrections de la boucle de variation par seconde
→ Temps de cycle 10ms TBC

9.3.8. gestion du réseau

Le manager de réseau DC10 aura pour tâche d'échanger périodiquement les données procédés (courant, consigne) et de gérer une table d'échanges composée de la liste des variables réseau du système et de leur temps de rafraîchissement.

DC10 utilisera la fenêtre de temps réservée aux messages apériodiques afin d'assurer le transfert des variables de sa liste dans le temps paramétré

La description de la table et de sa gestion seront décrit dans la spécification logicielle de DC10

9.3.9. Variables réseau

Numéro variable	description	Type données (référence langage C)	Taille (octets)	Lecture/ écriture
1	Consigne moteur 0 → OFF, 255 → ON, valeur intermédiaire = consigne PWM	Unsigned char	1	écriture
2	Courant Moteur Valeur Brut Capteur 10bit 0 à 1023	Unsigned int	2	lecture
3	Consigne Sens moteur → 0 Normal 1 arrière	bool	1	écriture
4	IHM sens moteur → 0 Normal 1 arrière	bool	1	lecture
5	Pédalage actif → 0 arrêt 1 Pédalage (capteur pédalage DC24)	bool	1	lecture
6	IHM accélération 0 à 255 (utilisation TBD)	Unsigned char	1	lecture

7	Capteur de pédalage DC22 Valeur nulle pour un sens de rotation et nombre de fronts reçu par le capteur de pédalage en 0,5s. Valeur saturée à 255.	Unsigned char	1	lecture
8	Valeur Brut rotation roue en nombre de fronts de la phase A en 0,5s max 255 équivalent de 255Hz (2 fronts par période)	Unsigned char	1	lecture
9	Freinage 0 inactif 1 Actif	bool	1	lecture
10	Éclairage Avant 0 Off 1 On	bool	1	écriture
11	Éclairage Arrière 0 Off 1 On	bool	1	écriture
12	Éclairage clignotant droit 0 Off 1 On	bool	1	écriture
13	Éclairage clignotant droit 0 Off 1 On	bool	1	écriture
14	Bouton action. 8 valeurs B0 à B7 → 0 relâché, 1 Appuyé	Unsigned char	1	lecture
15	Valeur Batterie Brut Capteur 10bit 0 à 2047	Unsigned int	2	lecture
16				
64	Variable complexe RUF	Meta donnée		
…….				
126	Variable complexe RUF	Meta donnée	6 Max	Ecriture/ lecture
127	Non utilisée (évite le déclenchement par un parasite (1 bit à 0 suffit)			

Seuil batterie

Moteur OFF et in-activable si la tension batterie est inférieure au seuil

Information Batterie

B15 = 0 Batterie OK moteur activable ou activé
B15 = 1 Batterie < Seuil et moteur arrêté
B0 à B14 valeur tension batterie en 1/10V

Commande réseau

1 → RUN réseau

2 → STOP réseau
3 → ERROR réseau
4 → INIT réseau

RPM Moteur

200 RPM est équivalent à 25 Km/h pour une roue de 26 pouces.

Méta Donnée

Elle ne sont pas décrites aujourd'hui. Elles seront utilisées pour des esclaves plus intelligents, dans un avenir plus ou moins lointain.
Elles ont pour objet de permettre l'échange de données plus conséquente en terme de taille ou en structure.
Elle serviront d'encapsulation à d'autre protocole éventuel.

Bien sur elle n'auront d'utilité qu'avec des contrôleurs susceptibles de les gérer

C'est la porte ouverte vers de nouveaux esclaves et contrôleurs avancés

10. DC17 Capteur physique

10.1. DC2A Capteur de Vitesse

TBD

10.2. DC2B Capteur de Freinage

TBD

10.3. DC2C Capteur de pédalage

TBD

10.4. DC2D Capteur d’accélération

TBD