MINISTÈRE
DE L'ÉDUCATION
NATIONALE
ET DE LA JEUNESSE
*Liberté*
*Égalité*
*Fraternité*

**EAE SIN 3**

**SESSION 2024**

# AGRÉGATION
# CONCOURS EXTERNE

## Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

## Option : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR ET INGÉNIERIE INFORMATIQUE

## CONCEPTION PRÉLIMINAIRE D'UN SYSTÈME, D'UN PROCÉDÉ OU D'UNE ORGANISATION

Durée : 6 heures

*Calculatrice autorisée selon les modalités de la circulaire du 17 juin 2021 publiée au BOEN du 29 juillet 2021.*

*L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.*

*Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.*

*Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.*

**NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.**
**Le fait de rendre une copie blanche est éliminatoire**

**Tournez la page S.V.P.**

**RECTIFICATIF**

- Les questions 8 et 9 sont annulées. Les candidats n'ont pas à les traiter.

- Page 11

  Question 11 (a) pour remplir le circuit, les candidats doivent préalablement recopier la figure 10 sur leur copie (pas de document réponse pour cette question).

- Page 38 - E Fichiers

  Au lieu de :

  - Fichier Gestion.js en fig ; 39

  Lire :

  - Fichier Gestion.js en fig ; 42

Ce sujet est composé de 46 pages :

— Un dossier questionnement de la page 3 à la page 30 ;

1. Présentation : page 3 à 4
2. Partie 1 : page 5 à 12
3. Partie 2 : page 13 à 19
4. Partie 3 : page 20 à 25
5. Partie 4 : page 26 à 30.

— des annexes techniques à partir de la page 31.
  — Schéma de principe d'une borne
  — Rapport cyclique EVSE
  — Courant de charge d'une borne
  — Protocole Modbus
  — Orno WE514 description
  — Fonction Modbus
  — Trame Modbus Maitre
  — Trame Modbus Esclave
  — Dialogue Modbus
  — Documentation OrnoWE514
  — Trame Orno WE514 Requete Réponse
  — Registres internes Orno WE514
  — Contacteur modbus
  — Organisation des ressources logicielles
  — Fichier Borne.js
  — Fichier Contacteur.js
  — Fichier Mesureur.js
  — Fichier Gestion.js
  — Programme Principal
  — Fichier EventPortSerie.js
— des documents réponses

Conseils aux candidats :

— Les deux premières parties du questionnement ne sont pas indépendantes ;
— Les parties 3 et 4 sont chacune indépendantes. Cependant, elles partagent un ensemble d'hypothèses présentées au démarrage de la partie 3 page 20 ;
— Un parcours attentif de l'ensemble du document est conseillé avant de composer.

# Table des matières





**Tournez la page S.V.P.**

# Etude de la réalisation et de la gestion d'une ou de plusieurs bornes de recharge pour véhicules électriques

Les véhicules électriques présentent de nombreux atouts. Ils permettent une réduction significative du $CO_2$ et des polluants émis lors de leur utilisation, excepté le système de freinage. Ils apportent également une solution économique compte-tenu du prix des énergies fossiles.

Ces véhicules sont en pleine expansion et séduisent de plus en plus d'automobilistes : à titre d'exemple, d'après le baromètre des immatriculations de aout 2023 publié par l'Avere France (association nationale pour le développement de la mobilité électrique), 19.657 véhicules particuliers électriques ont été immatriculés en aout 2023, ce qui correspond à 17.3% du total des véhicules particuliers immatriculés. Depuis janvier 2023, les véhicules électriques représentent 15.2% des immatriculations des véhicules particuliers.

La recharge de ces véhicules est un problème concret pour tout propriétaire d'un véhicule électrique. Plusieurs systèmes de recharge sont possibles (points de recharge public, recharge à domicile, au travail). Toujours d'après l'Avere France, 90% des usagers des véhicules électriques et hybrides rechargeables se branchent à leur domicile ou sur le lieu de leur entreprise. De plus, une voiture reste en moyenne connectée 90% du temps.

FIGURE 1 – Véhicule électrique à la recharge sur une borne de recharge individuelle.

Le sujet traite de quatre parties en lien avec la réalisation et l'utilisation d'une ou de plusieurs bornes de recharge pour véhicules électriques. Il est divisé en quatre grandes parties.

La première partie est consacrée à l'analyse de la constitution d'une borne de recharge reliée à un réseau de type `Modbus`. On analyse le dialogue entre la centrale de gestion et les



**Tournez la page S.V.P.**

périphériques de type `Modbus`. Le questionnement portera sur l'analyse d'une trame avec son code de validité puis l'élaboration d'une classe permettant de produire celle-ci.

Dans la seconde partie, nous analysons l'organisation d'une configuration à plusieurs bornes, le dispositif sera capable d'implémenter plusieurs bornes et d'opérer un dialogue permanent pour stocker les différentes mesures disponibles sur un serveur API.

Dans la troisième partie, on considère un parking (d'une entreprise ou public) constitué de plusieurs bornes de recharge de même puissance et un ensemble de demandes de charge, chacune liée à un véhicule électrique. On met en place un ensemble de fonctionnalités en Python qui permet d'associer les véhicules à des emplacements de parking. Notamment, on développe un algorithme efficace et optimal en nombre de places pour associer les véhicules aux places de parking.

FIGURE 2 – Bornes de recharges associées à plusieurs places de parking.

Dans la dernière partie, on se place toujours dans le cas d'un parking constitué de plusieurs bornes de recharge de même puissance. Cependant, l'allocation des véhicules aux places est donnée, mais la puissance totale de l'installation ne permet pas nécessairement de charger tous les véhicules simultanément. Dans un premier temps, on développe des outils algorithmiques pour décider des instants de charge de chaque véhicule de sorte à maximiser le nombre de véhicules chargés. Dans un second temps, on met en place des processus de synchronisation liés à l'entrée et la sortie de la station de recharge.

# 1 Analyse d'une borne

L'objectif de cette partie est d'analyser la constitution d'une borne de recharge reliée à un réseau de type `Modbus`.

---

La figure 3 montre l'organisation. Nous avons une unité de traitement déportée qui dialogue avec la borne se situant dans le parking au plus prés des véhicules à recharger.

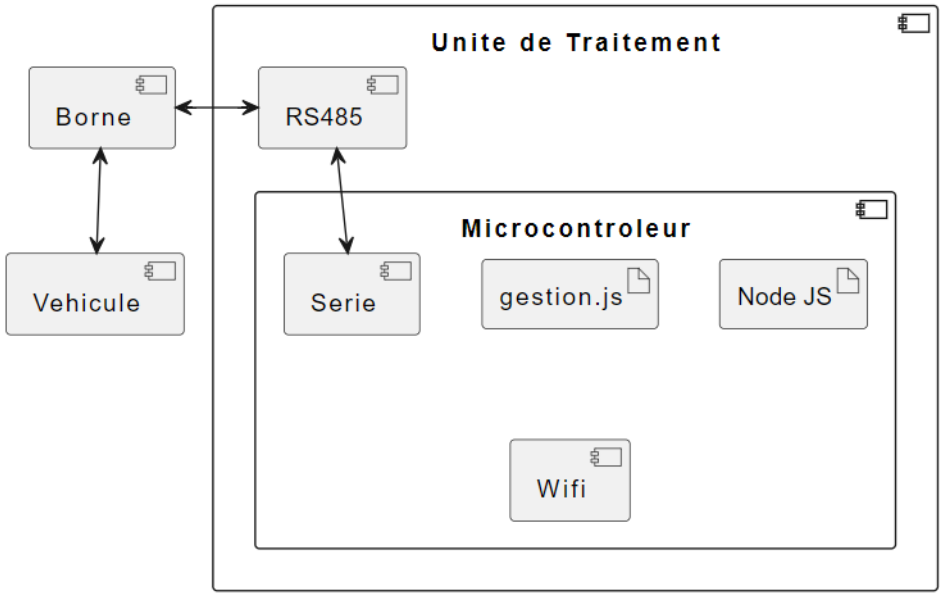



FIGURE 3 – Organisation de l'installation.

La borne est constituée de plusieurs éléments, comme montre dans la Figure 4. Elle est capable :

— de mettre en fonctionnement la recharge ;
— de mesurer la tension, l'intensité et la puissance délivrée au véhicule ;
— d'afficher différentes informations et de fixer la consigne maximale de courant.

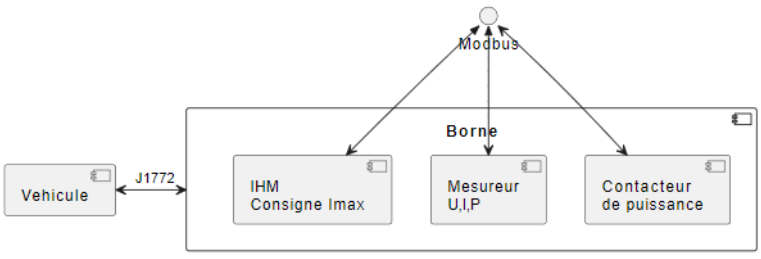



FIGURE 4 – Organisation d'une borne.



**Tournez la page S.V.P.**

## 1.1 Protocole J1772

Le dialogue entre la borne et le véhicule électrique utilise la norme J1772 comme montre dans la Figure 5.

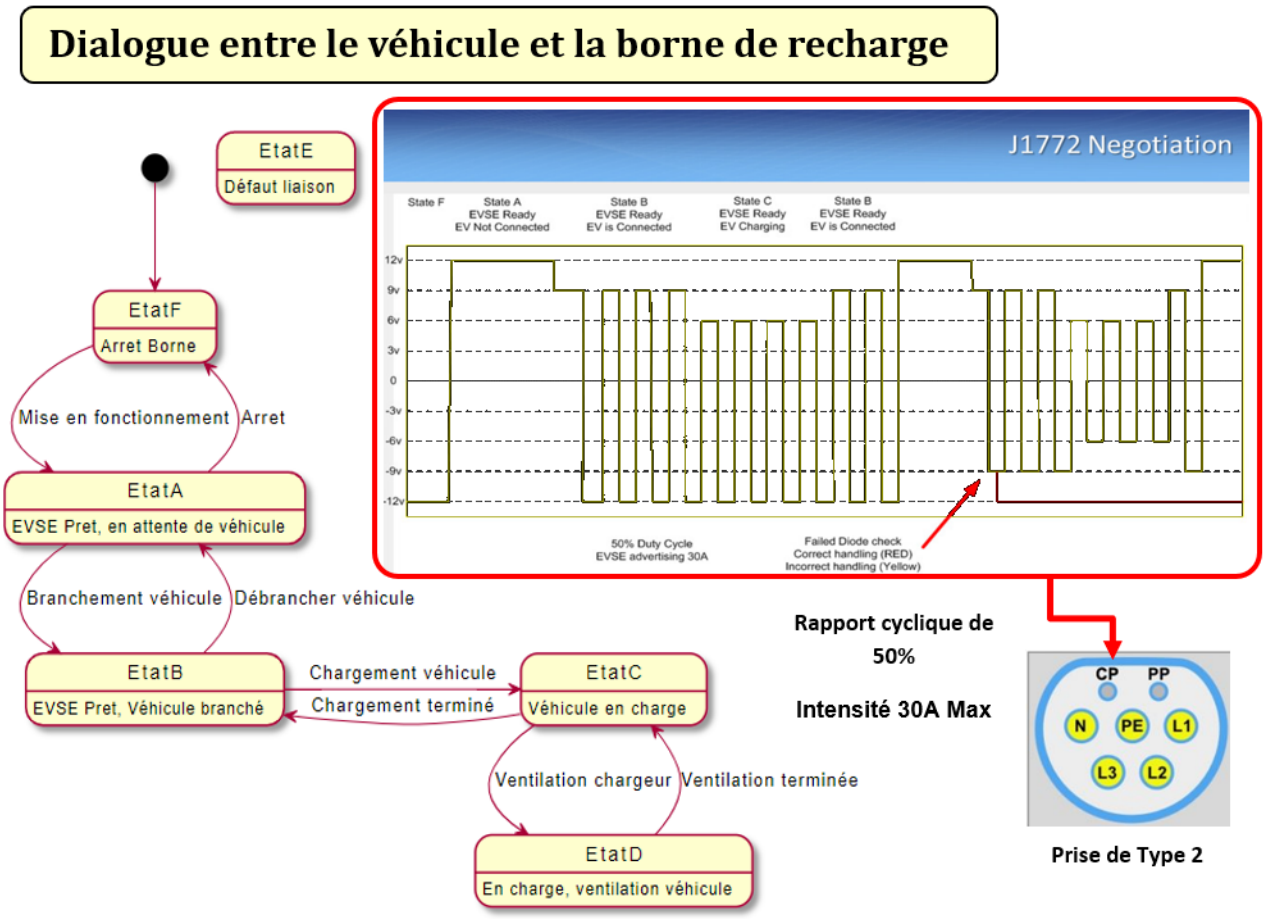



FIGURE 5 – Norme J1772

La prise normalisée de `type 2` (UE depuis 2013) permet une connexion monophasée ou triphasée au fournisseur d'énergie et d'imposer la consigne de chargement max et de connaître l'état du véhicule.

Descriptions des différentes broches :

— CP (Control Pilot) permet de dialoguer avec le véhicule électrique ;
— PP (Proximity Pilot) ;
— PE (Prise de terre) ;
— L1, L2 et L3 Phase 1 à 3 ;
— N (Neutre).

Dans la représentation temporelle nous pouvons observer la tension à la borne `CP`. La borne fournit deux tensions possibles

— -12V afin d'indiquer qu'elle n'est pas disponible ;

— +12V afin d'indiquer qu'elle est prête.

En retour lorsque le véhicule est branché, celui-ci consomme de l'énergie sur la borne CP afin de faire baisser la tension à + 9V.

La borne mesure cette tension et peut en déduire que le véhicule est branché. La consigne de courant max que peut consommer le chargeur du véhicule est fixé par un signal périodique de fréquence 1khz et de rapport cyclique variable.

**Question 1 :** Déterminer les équations permettant de définir le rapport cyclique par rapport au courant de consigne.

**Question 2 :** Proposer un algorithme pour déterminer le rapport cyclique en fonction de la valeur de consigne de courant en respectant la norme.

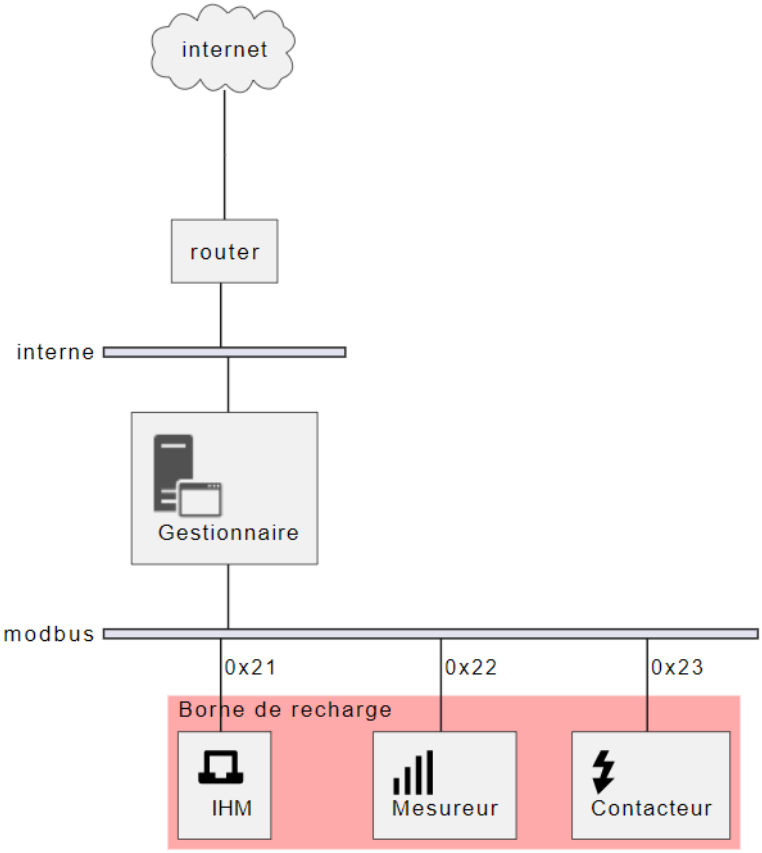



FIGURE 6 – Installation

La figure 6 propose l'organisation d'une borne connectée à un gestionnaire.

**Question 3 :** Justifier l'architecture proposée ?



**Tournez la page S.V.P.**

## 1.2 Protocole Modbus

### 1.2.1 Analyse du Mesureur

Le module ORNO514 est un périphérique communiquant utilisant le protocole Modbus. Soit la trame suivante sur le réseau modbus : 0x05 0x03 0x01 0x31 0x00 0x01 0xD5 0xBD

**Question 4 :** Analyser la requête que produit le gestionnaire sur le bus RS485 en complétant le tableau dans le document réponse.

Analyse de la réponse du module Mesureur ORNO.

| Adresse | Fonction | Donnees | CRC |
|---|---|---|---|
| 0x05 | 0x03 | 0x02 0x54 0x04 | 0x--0x-- |

FIGURE 7 – Trame réponse du Mesureur

**Question 5 :** Déterminer la valeur de la grandeur mesurée de la trame réponse de la Fig.7.Indiquer l'unité.

### 1.2.2 Analyse gestionnaire

Cette proposition va permette de produire la trame Modbus nécessaire à la communication entre le périphérique et le gestionnaire. Certaines classes sont incomplètes. Les classes Contacteur, IHM et Mesureur produisent une trame Modbus sans les deux derniers Octets, le calcul du CRC sera fait par la classe Borne. La figure 8 montre le diagramme de classes.

**Question 6 :** Justifier les relations de type 1..1 entre les classes.

L'unité de traitement utilise le moteur NodeJS avec des fichiers élaborés en Javascript.

### 1.2.3 Analyse de la Classe Contacteur

```
class Contacteur{
    constructor(adresseRelais=20){
        this._adresseRelais=adresseRelais
        this._etat=false
    }
    on(){..}
    off(){..}
}
```

La méthode on() permet de produire la trame permettant de rendre le contacteur actif et la méthode off() permet de produire la trame permettant de rendre le contacteur inactif.

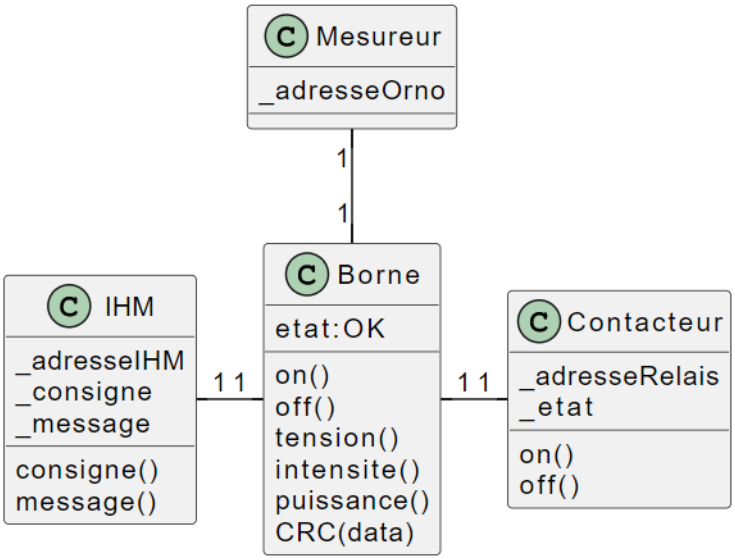



FIGURE 8 – Diagramme de classes

**Question 7 :** Compléter l'écriture des deux méthodes se trouvant dans le document réponse. Le relais utilisé sera le relais référencé 0.

## 1.3 Le Contrôle de Redondance Cyclique (CRC)

Le Contrôle de Redondance Cyclique (CRC) est un code de détection d'erreur basé sur la division polynomiale. Le champ de CRC du ModBus est constitué de deux octets contenant une valeur binaire de 16 bits (CRC16). La valeur CRC16 est calculée par le dispositif transmetteur, qui l'ajoute au message ou l'octet de poids faible sera transmis en premier, suivi de l'octet de poids fort.

Le flux de calcul de la CRC16 est illustré dans la Figure 9, ou XOR correspond au exclusif ou, N au nombre de bits d'information à tranmettre et POLY au polynôme de calcul du CRC16 qui est 1010 0000 0000 0001 (Polynôme générateur = $1+x^2+x^{15}+x^{16}$).

### 1.3.1 L'algorithme

**Question 8 :** Le code partiel d'une fonction en C permettant de calculer le CRC sur 16 bits est donné ci-dessous. La variable `data` contient le message dont le CRC doit être calculé et la variable `len` donne la taille du message en octets. Remplir dans le document réponse, le code de la fonction afin de calculer le CRC sur 16 bits du message sauvegardé dans la variable `data`.

**Question 9 :** On suppose la requête et la réponse sur le bus RS485 montrées dans le diagramme temporel suivant :



**Tournez la page S.V.P.**

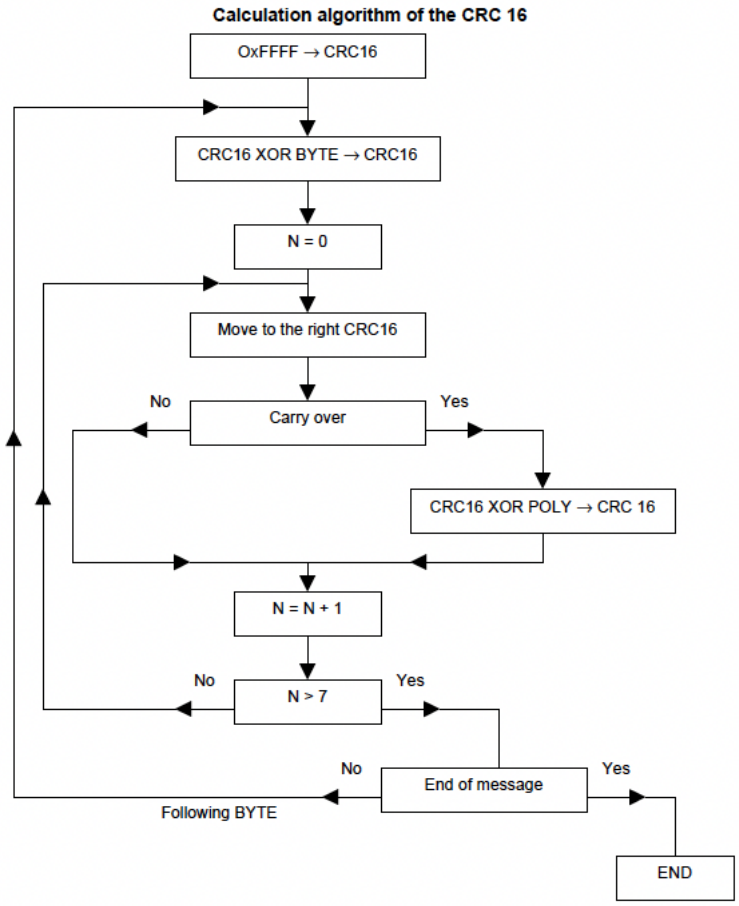
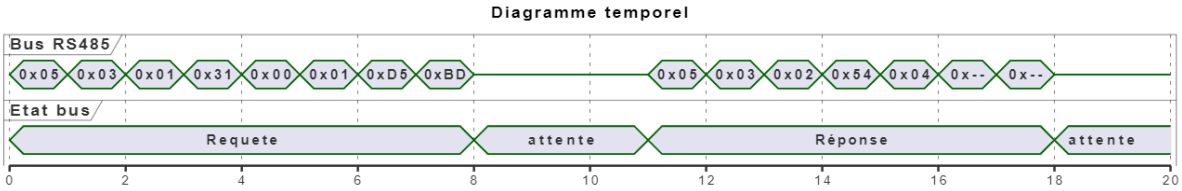



FIGURE 9 – Algorithme de calcul du CRC16.




(a) Après le traitement des trois premiers octets de la réponse à la requête de ce diagramme temporel, la variable CRC a la valeur 0xF0E0. Calculer la valeur finale du CRC et montrer les étapes du calcul.

(b) Remplir le diagramme temporel avec le code CRC calculé.

### 1.3.2 Implémentation matérielle

Dans cette partie, on souhaite explorer les différents matériels dédiés au calcul du CRC16.

**Question 10 :** L'implémentation en série est l'approche la plus simple dans les méthodes d'implémentation matérielle du calcul CRC.

(a) Dessiner une implémentation en série basée sur un registre à décalage à rétroaction linéaire (linear feedback shift register - LFSR).

(b) Donner l'équation qui calcule combien de cycles d'horloge sont nécessaires pour le calcul du CRC16 dans une implémentation en série ?

**Question 11 :** Une implémentation en série peut parfois s'avérer inadéquat en termes de débit. Cette limitation a conduit à plusieurs autres études qui se sont concentrées sur les implémentations parallèles. La méthode de parallélisation de l'implémentation du CRC consiste à dérouler le circuit série. Supposons que vous disposiez d'un circuit combinatoire capable de calculer le code CRC16 sur 8 bits en parallèle, appelé `parallel_CRC`, ce qui est illustré dans la Figure 10. Les entrées du circuit `parallel_CRC` sont `Data` qui sont codées sur un octet, et `CRC16_old` et `CRC16_new` qui sont codées sur deux octets. Les entrées du circuit d'implémentation parallèle sont :

— Les données `data` qui sont codées sur un octet,
— Un signal `d_valid` qui vaut 1 pour signaler au circuit chaque fois qu'un nouvel octet de donnée est valable,
— Un signal `init_comp` qui vaut 1 au début pour un cycle d'horloge pour signaler au circuit la demande du calcul CRC16,
— `len` code sur un octet qui donne la longueur du message.
— `clk` qui est l'horloge.
— `rst` qui est le reset.

La sortie du circuit est la valeur du CRC16 (`CRC16`) et un signal `CRC16_valid` pour signaler que le calcul a terminé.

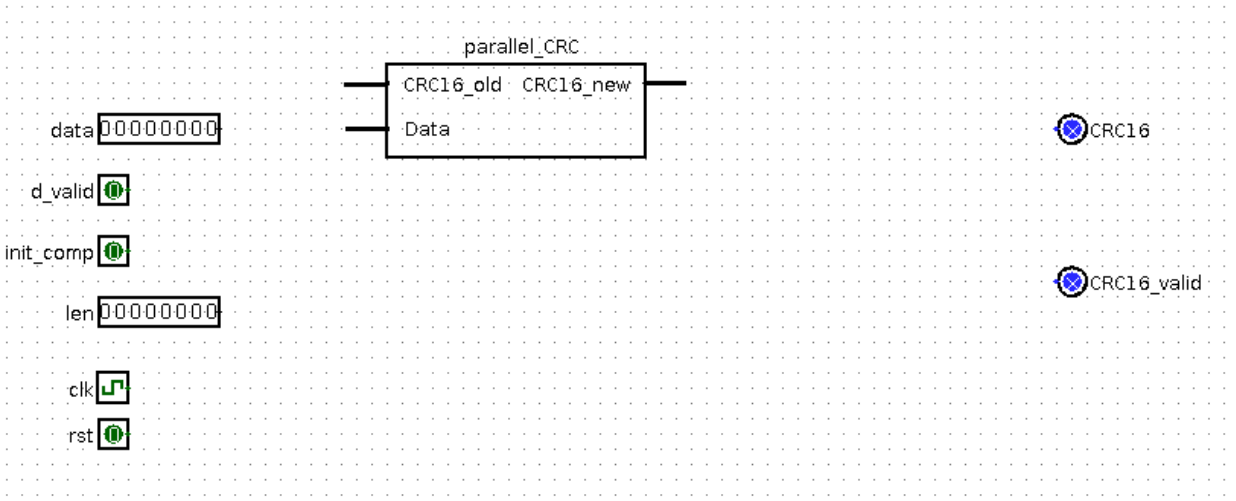



FIGURE 10 – Implémentation en parallèle

(a) Remplir le circuit en utilisant des multiplexeurs, des registres, des portes et des composants arithmétiques, afin de pouvoir calculer le CRC16 d'une message, quelle que soit la longueur.

(b) Donner l'équation qui calcule le nombre de cycles d'horloge nécessaires pour le calcul du CRC16 dans une implémentation de type parallèle ?



**Tournez la page S.V.P.**

**Question 12 :** Nous sommes intéressés par le gain que nous pouvons obtenir par une implémentation parallèle par rapport à une implémentation en série pour le calcul CRC16. Supossons le message 0x05 0x03 0x2 0x54 0x04.

(a) Combien de cycles d'horloge sont nécessaires pour le calcul du CRC16 pour l'implémentation en série ?

(b) Combien de cycles d'horloge sont nécessaires pour le calcul du CRC16 pour l'implémentation en paralléle ?

(c) Quelle réduction du temps de calcul (en pourcentage) l'implémentation en parallèle a-t-elle permit d'obtenir par rapport à l'implémentation en série ?

# 2 Analyse de plusieurs bornes

L'objectif de cette partie est d'analyser l'organisation d'une configuration à plusieurs bornes de type `Modbus`.

## 2.1 Organisation matériels

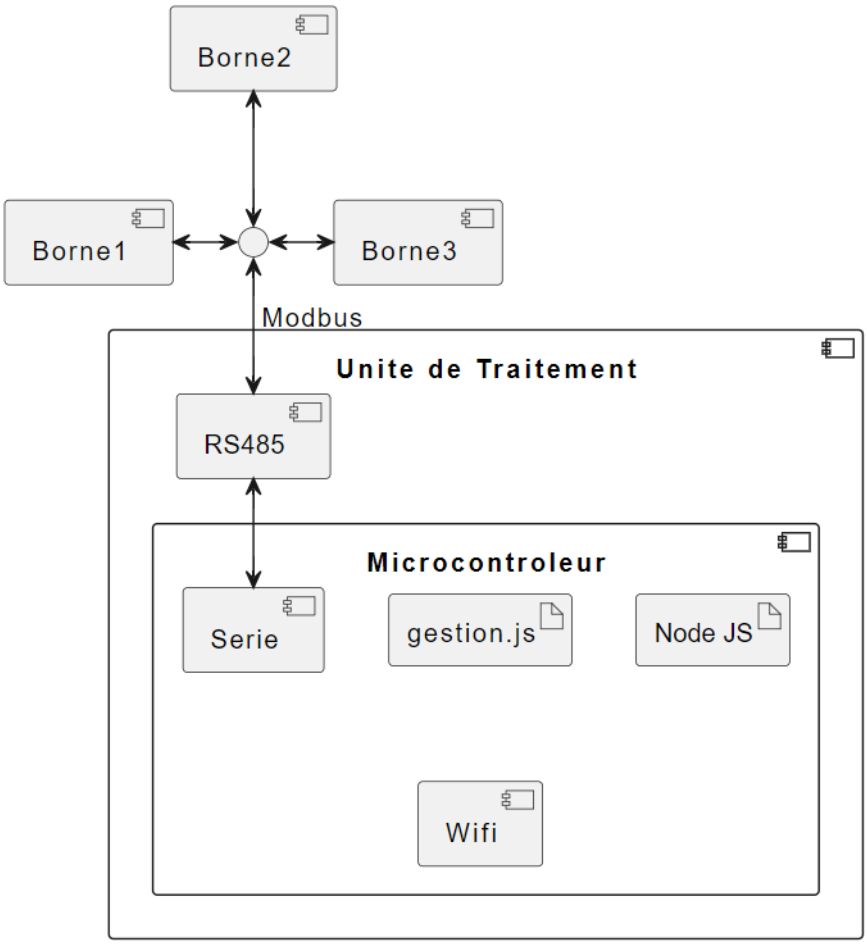



FIGURE 11 – Organisation à plusieurs bornes

**Question 13 :** Compléter le diagramme de classe se trouvant dans le document réponse, il faut déterminer les relations entre les classes en les justifiants.

(a) Gestionnaire ;

(b) Borne ;

(c) Serie (celle-ci gère le port série asynchrone).

*Voici un extrait du programme de gestion de l'unité de traitement.*



**Tournez la page S.V.P.**

```
import { Borne } from "./modules/borne.js"
import {SerialPort} from "serialport"

class Gestionnaire{
    constructor(){
        this.borne1=new Borne(0x11,0x21,0x31)
        this.borne2=new Borne(0x12,0x22,0x32)
        this.borne3=new Borne(0x13,0x23,0x33)
        this.borne4=new Borne(0x05,0x10,0x50)
        this.portCom="COM3"
        this.port=this.initSerie()
    }
    // Methodes pour le dialogue avec le port Serie
    initSerie(){..}
    write(data){..}
    read(){..}
    // Methode realisant la gestion des differentes bornes
    gestion(){..}
}
const gestion=new Gestionnaire()
```

On trouve la création de 4 bornes de recharge nommées borne1 à borne4, on instancie donc 4 objets de type borne avec différentes valeurs.

**Question 14 :** Indiquer la signification des trois valeurs passées en paramètres à l'instanciation d'un objet de type Borne.

**Question 15 :** Proposer une modification du programme afin de définir le port de communication série en paramètre à l'instanciation de l'objet gestion.

Maintenant nous allons exécuter le programme suivant :

```
// INSTANCIATION DE L'OBJET
const gestion=new Gestionnaire()

//console.log() permet d'afficher un message dans la console.
console.log("Production de Trames")
console.log("Trame pour Relais On :",gestion.borne1.actif())
console.log("Trame pour Lecture Tension :",gestion.borne2.tension())
console.log("Trame pour Lecture Intensite :",gestion.borne3.intensite())
console.log("Trame pour Lecture Puissance :",gestion.borne4.puissance())
```

Après exécution du programme on observe la réponse(figure 12).

**Question 16 :** Vérifier la validation des trames sans tenir compte du CRC.

## 2.2 Méthode puissance() et ajouterCRC() de la classe Borne

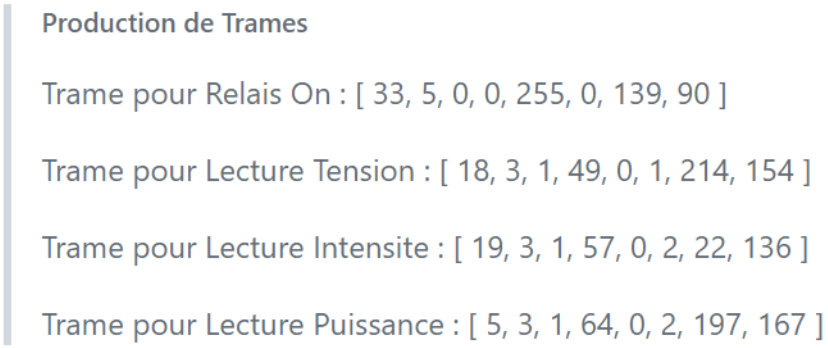



FIGURE 12 – Réponse dans la console

```
// Extrait de la classe Borne
// On trouve les deux methodes suivantes:

puissance(){
      return  this.ajouterCRC(this._Mesureur.powerActiveV1())
    }

ajouterCRC(data){
      let trame=data
      let CRC=  this.CRC(trame)
      trame.push(Math.trunc(CRC%256))
      trame.push(Math.trunc(CRC/256))
      return trame
      }
```

**Question 17 :** (a) Expliquer l'écriture de la méthode puissance() ;

(b) Pour quelle raison avons nous CRC%256 et CRC/256 dans la méthode ajouterCRC() ?

## 2.3 Analyse temporelle d'une communication Modbus

Reprenons l'exemple pour une requête de demande de tension (figure 13).

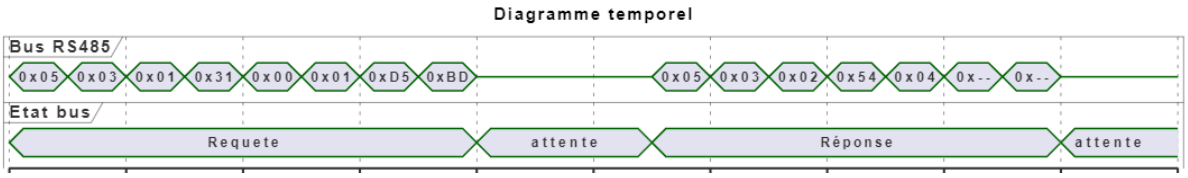



FIGURE 13 – Requete Modbus



**Tournez la page S.V.P.**

Le gestionnaire fait sa requête en envoyant sa trame, la norme précise qu'il doit exister une durée de **silence** minimale de 3.5 caractères entre une requête et la réponse d'un périphérique.

**Question 18 :** Déterminer la durée minimale de **silence**.

Notre système est basé sur un protocole **Requête-Réponse**, ensuite on passe à la requête suivante afin de définir un cycle. Ce cycle permet de maintenir à jour les différentes mesures.

**Question 19 :** Si un périphérique ne répond pas que se passe-t-il ?

## 2.4 Exploitation des mesures

Nous souhaitons que les résultats de mesures soient disponibles sur un serveur API de type CRUD pour alimenter des bases de données et exploiter celles-ci via des clients Web. Ce serveur permet de lire, stocker les différentes mesures d'un équipement de plusieurs bornes de recharge pour véhicule électrique. La solution envisagée se trouve en Fig 14.


FIGURE 14 – Serveur

**Question 20 :** Que signifie CRUD ?

## 2.5 Machine d'état

Dans une organisation à plusieurs bornes il est nécessaire de gérer l'ensemble par cycle de mesure. La solution envisagée est d'organiser la gestion de cycle par une machine d'état.

Rôle de la machine d'état :

— Elle doit gérer le séquencement de l'ensemble des transactions pour l'émission et la réception de trame au format Modbus.
— Elle est capable de traiter un cycle de plusieurs trames formatées Modbus.
— Les différentes réponses sont toutes enregistrées dans un buffer.
— Elle est capable de traiter un dialogue périodique avec un ou des périphériques Modbus.
— La communication avec le port Série s'effectue par événement.

### 2.5.1 Diagramme de séquence

Pour illustrer le fonctionnement de l'ensemble, on s'appuie sur le diagramme de séquence de la Figure 15.

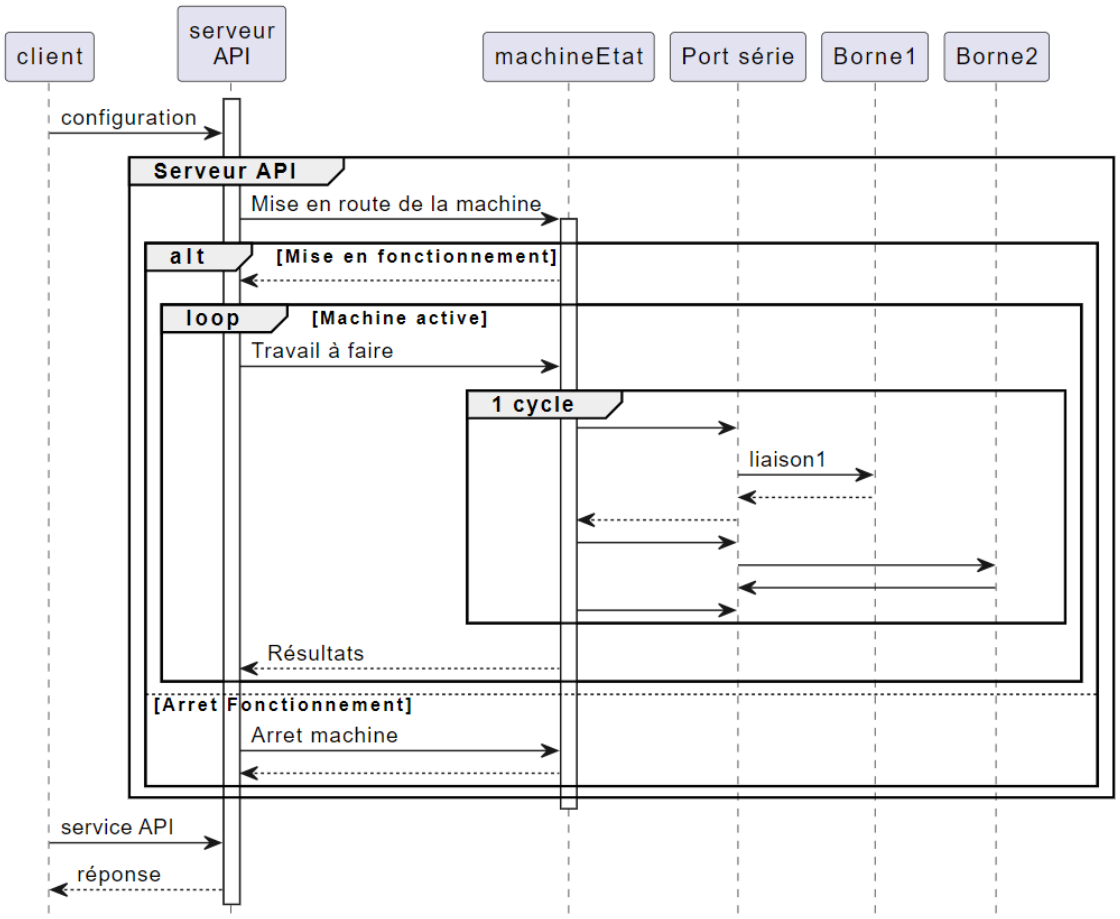



FIGURE 15 – Diagramme de séquence

**Question 21 :** (a) Dans l'organisation proposée, comment démarre un cycle ?

(b) Le mesureur de la borne1 est à l'adresse 0x05, le mesureur de la borne2 est à l'adresse 0x06, en vous aidant des résultats fournit par la machine d'état pour un cycle de fonctionnement (fig 16), indiquer le véhicule déjà chargé ?

### 2.5.2 Structure de la machine d'état

**Question 22 :** Indiquer le rôle de l'état TimeOut dans la machine d'état de la Figure 17 ?

**Question 23 :** Compléter dans le document réponse le tableau pour un cycle émettant sur le réseau Modbus, les trois trames sont définies dans la Figure18).

**Question 24 :** Déterminer la durée du time Out sachant que l'échange possible de donnée sera au maximum de 50 Octets.

**Question 25 :** D'après l'organisation proposée de quelle manière peut-on programmer la machine d'état ?



**Tournez la page S.V.P.**

| (index) | Trame requete | réponse |
|---|---|---|
| Trame0 | [32, 5, 0, 0, 255, 0, 138, 139] | [ 20 05 00 00 ff 00 8a 8b] |
| Trame1 | [5, 3, 1, 49, 0, 1, 213, 189] | [05 03 02 53 f2 f4 f1] |
| Trame2 | [5, 3, 1, 57, 0, 2, 20, 126] | [05 03 04 00 00 00 00 bf f3 ] |
| Trame3 | [6, 3, 1, 57, 0, 2, 20, 77] | [06 03 04 00 01 9a 28 8c f3] |
| Trame4 | [5, 3, 1, 64, 0, 2, 197, 167] | [05 03 04 00 00 00 00 bf f3] |
| Trame5 | [6, 3, 1, 64, 0, 2, 197, 148] | [06 03 04 00 23 28 05 4c f0] |
| Trame6 | [32, 5, 0, 0, 0, 0, 203, 123] | [20 05 00 00 00 00 cb 7b] |

FIGURE 16 – Résultats au bout d'un cycle machine

### 2.5.3 Programmation événementielle

**Question 26 :** Indiquer l'avantage d'utiliser cette technologie ? Justifier son utilisation dans ce projet.

Nous allons décrire le fonctionnement de la communication série par événement.

```
/* Pour definir un evenement il faut:

1) Importer le module EventEmitter de node:events
--> import { EventEmitter, errorMonitor } from "node:events";

2) Declarer un objet de type EventEmitter
--> const eventModbus=new EventEmitter()

3) Mettre en place les ecouteurs d'evenements,
Il suffit d'utiliser la methode on() et d'indiquer le canal d'ecoute et
l'action a effectuer.
Exemple:
--> eventModbus.on("canal1",gestionCanal1) lorsqu'un evenement est emit
sur le "canal1" alors on execute la fonction gestionCanal1

4) Mettre en place les emetteurs:
Pour emettre un evenement, il suffit d'utiliser la methode emit(),
d'indiquer le "canal" et d'y adjoindre les donnees.
Exemple :
--> eventModbus.emit("Trame","Trame emise")
Ici on vient d'emettre sur le canal Trame la donnee "Trame emise"

Il est possible de definir plusieurs ecouteurs et emetteurs differents.
```

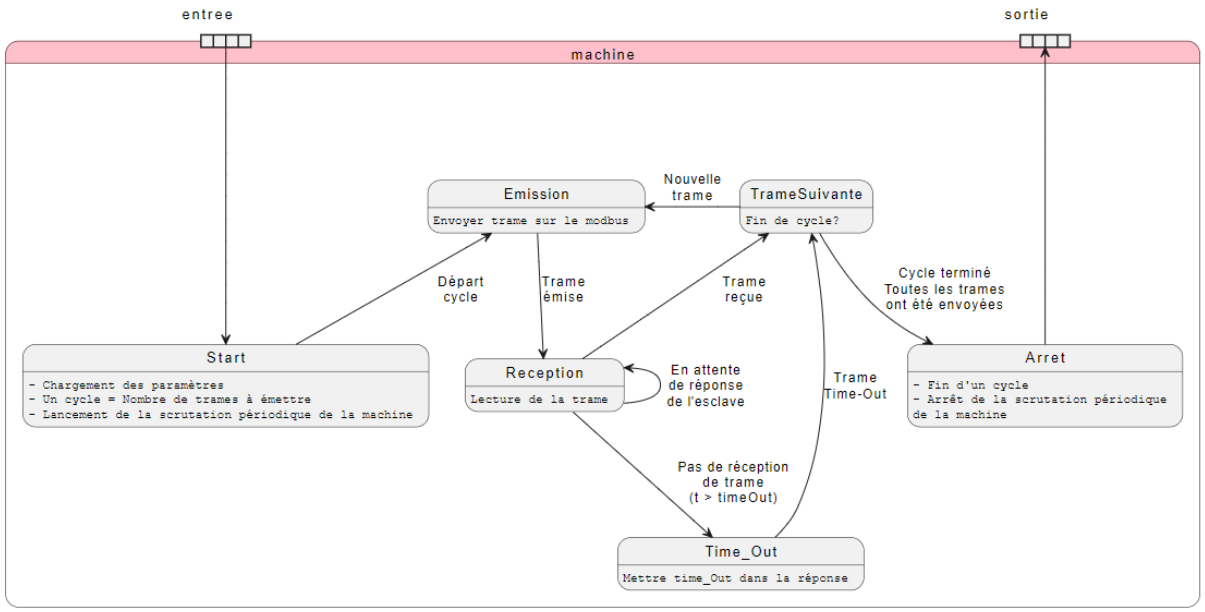



FIGURE 17 – Machine d'état

| (index) | Trame requete | réponse |
|---|---|---|
| Trame0 | [32, 5, 0, 0, 255, 0, 138, 139] | [ 20 05 00 00 ff 00 8a 8b] |
| Trame1 | [51, 3, 1, 49, 0, 1, 213, 189] | ne réponds pas |
| Trame2 | [5, 3, 1, 57, 0, 2, 20, 126] | [05 03 04 00 00 00 00 bf f3 ] |

FIGURE 18 – Analyse machine d'état

*/

**Question 27 :** En analysant le programme eventPortSerie.js se trouvant en annexe compléter dans le document réponse le tableau répertoriant les différents événements.

**Question 28 :** Comment utiliser le module eventPortSerie.js afin de transmettre et réceptionner des données ?



**Tournez la page S.V.P.**

# 3 Allocation des demandes aux chargeurs

L'objectif de cette partie est de développer un ensemble d'outils pour récupérer l'ensemble des données liées aux véhicules, et de proposer à l'avance si cela est possible une allocation des véhicules aux places de parking associées à des chargeurs.

---

On considère un ensemble $\mathcal{V} = \{1, \ldots, n\}$ de demandes associées à des véhicules électriques. On suppose que, dans la période considérée, il ne peut y avoir deux demandes pour un même véhicule. Ces véhicules sont à connecter à un ensemble de chargeurs $\mathcal{C} = \{1, \ldots, m\}$ identiques. A tout instant, un chargeur ne peut charger qu'un seul véhicule, et un véhicule ne peut être chargé que par un seul chargeur à la fois. De plus, chaque chargeur peut délivrer une puissance constante maximum de $w$ (en kW).

La demande de charge d'un véhicule $v \in \mathcal{V}$ est associée à une date d'arrivée $r_v$, de départ $d_v$ et à une énergie requise $e_v$ en kWh. Le véhicule $v$ occupe un chargeur et la place de parking associée durant tout l'intervalle $[r_v, d_v[$, même si il est complètement chargé avant $d_v$. De plus, le véhicule doit avoir récupéré si c'est possible l'énergie requise à son départ.

On suppose que la période de temps considérée est divisée en $\delta$ intervalles d'une durée $\mathcal{T}$ chacune et que toutes les dates et les durées considérées sont des multiples de $\mathcal{T}$. Dates et durées manipulées sont alors ramenées à des valeurs entières dans l'intervalle $[0, \ldots, \delta]$.

Ainsi, pour toute demande $v \in \mathcal{V}$,

— on note $\underline{r}_v = \lfloor \frac{r_v}{\mathcal{T}} \rfloor$ la date au plus tôt pour l'arrivée du véhicule dans la place associée au chargeur ;

— on note également $\overline{r}_v = \lceil \frac{r_v}{\mathcal{T}} \rceil$ la date considérée pour démarrer la charge du véhicule.

De la même manière, pour toute demande $v \in \mathcal{V}$,

— on note $\overline{d}_v = \lceil \frac{r_v}{\mathcal{T}} \rceil$ la date considérée pour le départ effectif du véhicule ;

— on note également $\underline{d}_v = \lfloor \frac{r_v}{\mathcal{T}} \rfloor$ la date maximum de fin de charge.

On rappelle que, pour tout réel $x$, $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à $x$ et $\lfloor x \rfloor$ le plus grand entier inférieur ou égal à $x$.

Enfin, on suppose que le temps de charge effectif $p_v$ associé à la demande $v$ vérifie $p_v = \lceil \frac{e_v}{w.\mathcal{T}} \rceil$. Le problème traité dans la première sous-partie est la récupération des données associées aux demandes, et la modélisation de l'allocation des véhicules aux chargeurs.

## 3.1 Gestion des liens entre chargeurs et demandes

On suppose dans cette partie que l'on travaille sur une durée de 24 heures avec une précision d'un quart d'heure. Ainsi, $\mathcal{T} = 900$ secondes et une journée est constituée de $\delta = 96$ quart d'heures.

Toute demande est un objet de la classe `Demande` en Python qui obéit à la déclaration suivante :

```
class Demande:
```

```
    def __init__(self,inf_arr,sup_arr,inf_dep,sup_dep,t_charge,im):
        self.inf_r = inf_arr
        self.sup_r = sup_arr
        self.inf_d = inf_dep
        self.sup_d = sup_dep
        self.p = t_charge
        self.imma = im
```

La classe `Demande` est constituée de 5 attributs entiers `inf_r`, `sup_r`, `inf_d`, `sup_d` et `p` correspondant respectivement aux valeurs $\underline{r}_v$, $\overline{r}_v$, $\underline{d}_v$, $\overline{d}_v$ et $p_v$ pour une demande $v$. Le dernier attribut `imma` correspond à la plaque d'immatriculation du véhicule sous la forme d'une chaîne de caractères.

Les demandes à considérer sont transmises au système sous la forme d'un fichier `xml` dont un exemple est représenté par la figure 19.

```
<BaseDemandes>
    <demande immatriculation="XW-540-FF" heureArrivee="10" minuteArrivee="50"
heureDepart="12"  minuteDepart="25"  energieRequise="50">
    </demande>
    <demande immatriculation="WK-975-XZ" heureArrivee="11" minuteArrivee="45"
heureDepart="19"  minuteDepart="40"  energieRequise="65">
    </demande>
    <demande immatriculation="PR-277-BA" heureArrivee="9" minuteArrivee="45"
heureDepart="14"  minuteDepart="22"  energieRequise="61">
    </demande>
</BaseDemandes>
```

FIGURE 19 – Exemple d'un fichier de demandes.

On souhaite stocker dans un dictionnaire l'ensemble des demandes avec pour clefs les immatriculations des véhicules concernés. De plus, les demandes stockées dans le dictionnaire doivent être réalisables :

— chaque demande doit pouvoir être réalisée entre les instants 0 et $\delta$, sachant que les heures et les minutes stockées dans le fichier `xml` ont des valeurs correctes, càd les heures sont comprises entre 0 et 23 et les minutes entre 0 et 59 ;

— le temps de charge est suffisant pour récupérer l'énergie requise par la demande.

Pour cela, on considère la fonction Python `dictionnaire_demandes` dont une définition partielle suit. Le paramètre `root` est la racine de l'arbre obtenu par l'appel du parser sur un fichier `xml` de demandes. Chacun de ses fils est associé à une demande. Les paramètres `delta`, `tau` et `w` correspondent respectivement aux valeurs $\delta$, $\tau$ et $w$.

```
import xml.etree.ElementTree as ET
import math as MT

def dictionnaire_demandes(root,delta,tau,w):
    dico_demandes = {}
    for child in root:
```



**Tournez la page S.V.P.**

```
            # recuperation des donnees d'une demande et conversions
            inst_a = Demande.time_to_secondes(int(child.get("heureArrivee")),
    int(child.get("minuteArrivee")))
            inst_d = Demande.time_to_secondes(int(child.get("heureDepart")),
    int(child.get("minuteDepart")))
            im = child.get("immatriculation")
            energie = int(child.get("energieRequise"))

            #calcul des attributs d'une demande
            (a)
            #test de la correction de la demande et creation
            if (b)...
            else:
                dico_demandes[im] = Demande(inf_arr,sup_arr,inf_dep,sup_dep,
    duree,im)
        return dico_demandes
```

On pourra également considérer les fonctions `floor(x)` et `ceil(x)` de la bibliothèque Python `math` qui retournent respectivement les entiers $\lfloor x \rfloor$ et $\lceil x \rceil$ pour toute valeur $x$.

**Question 29 :** (a) Calculer les valeurs `inf_arr`, `sup_arr`, `inf_dep`, `sup_dep` et `duree` correspondant respectivement aux valeurs $\underline{r}_v$, $\overline{r}_v$, $\underline{d}_v$, $\overline{d}_v$ et $p_v$ associés à la demande $v$ en cours (Partie (a) du programme Python) ;

(b) effectuer les tests nécessaires pour que la demande soit réalisable et la stocker dans le dictionnaire le cas échéant. Renvoyer un message d'erreur explicite en cas de problème.

**Question 30 :** On souhaite maintenant associer l'ensemble des demandes stockées dans le dictionnaire à $m$ places de parking associées chacune à un chargeur de puissance $w$. Pour cela, on définie la classe `Chargeur` de la manière suivante :

```
class Chargeur:
    def __init__(self):
        self.liste_demandes = []
```

Durant la période de temps considérée $[0, \delta]$, plusieurs demandes peuvent être associées à une seul place de parking (et donc à un seul chargeur) à condition qu'elles ne soient pas deux à deux simultanées. L'attribut `liste_demandes` permet ainsi de stocker la liste des demandes associées au chargeur considéré stockées selon l'attribut `inf_r` croissant.

(a) Soient deux intervalles non vides $[x_1, y_1[$ et $[x_2, y_2[$. Exprimer une condition nécessaire et suffisante simple sur les valeurs $x_1$, $x_2$, $y_1$ et $y_2$ pour que l'intersection de ces intervalles soit vide ;

(b) en déduire le code Python de la méthode `intervalles_disjoints(self,v)` de la classe `Demande` qui retourne `True` si la demande courante et la demande `v` peuvent partager le même chargeur.

**Question 31 :** Dans cette question, on souhaite rajouter une demande à un chargeur. On effectue cette insertion (si elle est possible) par la primitive `rajoute_demande(self,v)` de la classe `Chargeur` qui retourne `True` si et seulement si cette insertion est effectuée. Le code partiel de cette primitive suit :

```
def rajoute_demande(self,v):
    if (self.liste_demandes==[]):
        self.liste_demandes.append(v)
        v.lien_chargeur(self)
        return True
    tailleListe=len(self.liste_demandes)
    for i in range(tailleListe):
        if not v.intervalles_disjoints(self.liste_demandes[i]):
            return False
        #Si l'insertion est possible avant self.liste_demandes[i]
        (a)
            return True
    # Sortie de boucle
    (b)
```

(a) Donner le code manquant (a) qui teste si la demande `v` peut être rajoutée juste avant la demande `self.liste_demandes[i]` et effectue cette insertion correctement ;

(b) dans quel cas sort on de la boucle sans passer par un `return` ? En déduire le code (b) manquant ;

(c) quelle est la complexité de la primitive `rajoute_demande(self,v)` dans le meilleur et le pire des cas. Justifier votre réponse en exhibant un pire cas et un meilleur cas.

## 3.2 Attribution des places de parking aux demandes

On souhaite par la suite développer un algorithme qui attribue un ensemble de demandes stockées dans un dictionnaire `dicoDemandes` à un ensemble de chargeurs stockés dans une liste `listeChargeurs`. Le code de ce premier algorithme est le suivant :

```
def allocation_demande_chargeurs(dicoDemandes, listeChargeurs):
    resultat = []
    for d in dicoDemandes.values():
        for c in listeChargeurs:
            if (c.rajoute_demande(d)):
                break
        if (not d.chargeur):
            resultat.append(d)
    return resultat
```

On considère alors comme exemple les 6 demandes décrites dans la table 1 à allouer à 3 chargeurs. Ces 6 demandes sont stockées dans l'ordre de la table dans le dictionnaire des demandes `dicoDemandes`. Elles sont par la suite identifiées par leur immatriculation.



**Tournez la page S.V.P.**

| imma | inf_r | sup_r | inf_d | sup_d | p |
|---|---|---|---|---|---|
| 'A' | 0 | 0 | 3 | 3 | 3 |
| 'B' | 2 | 2 | 5 | 5 | 3 |
| 'C' | 1 | 1 | 5 | 6 | 3 |
| 'D' | 5 | 5 | 7 | 7 | 1 |
| 'E' | 4 | 4 | 7 | 7 | 2 |
| 'F' | 6 | 6 | 8 | 8 | 2 |

TABLE 1 – Description de 6 demandes stockées dans l'ordre de la table dans le dictionnaire des demandes `dicoDemandes`.

**Question 32 :** (a) Donner le résultat de la fonction `allocation_demande_chargeurs` pour les 6 demandes décrites dans la table 1 sur 3 chargeurs. Pour cela, donner pour chacun des chargeurs la liste des demandes associées. Est-ce que toutes les demandes ont pu être allouées à un chargeur ?

(b) Est-ce que il existe une allocation sur 3 chargeurs qui permette de satisfaire toutes ces demandes ? Justifier votre réponse.

**Question 33 :** Dans cette question, on souhaite déterminer le nombre minimum de chargeurs nécessaires $m^\star(\mathcal{D})$ pour traiter un ensemble des demandes $\mathcal{D}$. Pour cela, on associe à un ensemble de demandes $\mathcal{D}$ un graphe d'intervalles $G_\mathcal{D} = (V, E)$ definie de la manière suivante :

— Les sommets de $G_\mathcal{D}$ sont les demandes, soit $V = \mathcal{D}$ ;

— Pour tout couple $(u, v) \in \mathcal{D}^2$ avec $u \neq v$, l'arête $e = \{u, v\} \in E$ si $[\underline{r}_u, \bar{d}_u[ \cap [\underline{r}_v, \bar{d}_v[ \neq \emptyset$, autrement dit $u$ et $v$ ne peuvent être allouées au même chargeur (et à la même place de parking) compte tenu des contraintes temporelles.

Par exemple, si on considère les demandes des voitures $A$ et $B$, les deux intervalles associés sont respectivement $[\underline{r}_A, \bar{d}_A[ = [0, 3[$ et $[\underline{r}_B, \bar{d}_B[ = [2, 5[$. Leur intersection est non vide, ce qui signifie que les véhicules $A$ et $B$ ne peuvent se partager la même place de parking. L'arête $\{A, B\} \in E$.

Une coloration en $k$ couleurs des sommets consiste à déterminer la fonction $f : V \to \{0, \ldots, k-1\}$ telle que, pour toute arête $e = \{u, v\} \in E$, $f(u) \neq f(v)$. Le nombre minimum de couleurs nécessaires pour colorier le graphe $G_\mathcal{D}$ est noté $\chi(G_\mathcal{D})$.

(a) Construire le graphe d'intervalles $G_\mathcal{D}$ associé aux 6 commandes décrites par la table 1 ;

(b) quel est le nombre minimum de couleurs nécessaires pour obtenir une coloration de ce graphe ? Justifier votre réponse.

**Question 34 :** On se place dans le cas général. On suppose que les chargeurs sont notés $c_0, \ldots, c_{m-1}$.

(a) Démontrer que l'on une coloration en $k$ couleurs de $G_\mathcal{D}$ si et seulement si on peut allouer les demandes à $k$ chargeurs ;

(b) en déduire que, dans le cas général, le nombre minimum de chargeur $m^\star(\mathcal{D})$ vaut $\chi(G_\mathcal{D})$ ;

(c) en déduire pour l'exemple une allocation des demandes en $m^\star(\mathcal{D})$ chargeurs à partir d'une coloration du graphe $G_\mathcal{D}$ en $\chi(G_\mathcal{D})$ couleurs.

**Question 35 :** Soit $G = (V, E)$ un graphe non orienté quelconque. Un sous-graphe $H = (V', E')$ d'un graphe $G = (V, E)$ est un graphe tel que $V' \subseteq V$ et $E'$ est l'ensemble des arêtes $\{x, y\}$ de $E$ telles que $(x, y) \in V'^2$. Une clique de taille $w > 0$ de $G$ est un sous-graphe $H = (V', E')$ de $w$ sommets de $G$ complet, i.e. tout couple de sommets $(x, y) \in V'^2$ avec $x \neq y$ est relié par une arête $\{x, y\}$. On note par la suite $\omega(G)$ la taille maximale d'une clique de $G$.

Montrer que, pour tout graphe $G$, $\omega(G) \leq \chi(G)$.

On suppose dans les trois questions suivantes que les demandes sont stockées dans un dictionnaire en suivant les attributs `inf_r` croissants.

**Question 36 :** Dans cette question, on considère l'exemple donné par la table 1.

(a) Donner le résultat de l'exécution de la fonction `allocation_demande_chargeurs` pour ces 6 demandes pour 3 chargeurs. On rappelle que les demandes sont stockées dans un dictionnaire en suivant les attributs `inf_r` croissants ;

(b) que vaut $\omega(G_{\mathcal{D}})$ pour l'exemple ?

**Question 37 :** Dans cette question, on se place dans le cas général. Soit $p \geq 1$.

(a) Montrer que, si l'algorithme attribue une demande $d$ au chargeur $c_p$, alors le graphe $G_{\mathcal{D}}$ possède une clique de taille $p$ ;

(b) en déduire que, si l'algorithme alloue toutes les demandes, le nombre de chargeurs utilisés est minimal. Pour cela, on pourra démontrer que $\overline{m}$ le nombre de chargeurs alloués par l'algorithme vérifie $\overline{m} = m^\star(\mathcal{D}) = \chi(G_{\mathcal{D}}) = \omega(G_{\mathcal{D}})$.

**Question 38 :** On souhaite évaluer la complexité de `allocation_demande_chargeurs`. Pour cela, on pose $n = |\mathcal{D}|$ et $m$ le nombre de chargeurs.

(a) Quelle est la complexité nécessaire pour trier les demandes selon le champs `inf_r` croissant en effectuant un tri de comparaison ?

(b) Evaluer une borne supérieure de la complexité de `allocation_demande_chargeurs`. Justifier votre réponse ;

(c) Peut-on améliorer cette complexité ? Justifier votre réponse.



**Tournez la page S.V.P.**

# 4 Détermination des périodes de chargement des véhicules

Dans la dernière partie, on considère un parking constitué de plusieurs bornes de recharge de même puissance. L'allocation des véhicules aux places est fixée, mais la puissance totale de l'installation ne permet pas nécessairement de charger tous les véhicules simultanément. Dans un premier temps, on développe des outils algorithmique pour décider des instants de charge de chaque véhicule de sorte à maximiser le nombre de véhicules chargés. Dans un second temps, on met en place des processus de synchronisation liés à l'entrée et la sortie de la station de recharge.

---

## 4.1 Allocation des périodes de charges aux demandes

On suppose dans cette partie que les demandes de $\mathcal{D}$ ont été allouées à $m$ chargeurs $c_0, \ldots, c_{m-1}$. Les demandes sont numérotées $v_0, \ldots v_{n-1}$ de manière quelconque.

La période de temps considérée est divisée en $\delta$ intervalles d'une durée $\mathcal{T}$ chacune et que toutes les dates et les durées considérées sont des multiples de $\mathcal{T}$. Dates et durées manipulées sont alors ramenées à des valeurs entières dans l'intervalle $[0, \ldots, \delta]$.

Chaque chargeur délivre une puissance constante de $w$kW. Cependant, la puissance totale de l'installation est strictement inférieure à $m \times w$. Ainsi, on ne peut charger pendant tout intervalle de temps $[\alpha, (\alpha+1)[$ avec $\alpha \in \mathbf{N}$ qu'au plus $\widetilde{m} < m$ voitures simultanément. De plus, on considère que la décision de charger (ou de non charger) une voiture est effectuée aux instants $t = \alpha$ pour toute la durée de l'intervalle $[\alpha, \alpha+1[$.

Pour toute demande $v$, on rappelle que le véhicule associé doit être chargé si possible pendant $p_v$ intervalles de longueur 1 durant $[\overline{r}_v, \underline{d}_v[$.

Pour toute demande $v \in \mathcal{D}$ et tout entier $t \in \{0, \ldots, \delta - 1\}$, on pose $x_{v,t} = 1$ si le véhicule correspondant à $v$ est en charge pendant l'intervalle $[t, t+1[$. Dans le cas contraire, $x_{v,t} = 0$.

Le problème posé est de déterminer les valeurs $x_{v,t} \in \{0, 1\}$ telles que la solution obtenue vérifie l'ensemble des contraintes du système, et que le temps de charge global des véhicules soit maximum. La matrice de taille $|\mathcal{D}| \times \delta$ correspondant à l'ensemble de ces valeurs et est notée $x_{\star,\star}$.

**Question 39 :** On considère une solution réalisable au problème d'allocation de charge traité pour le problème exprimé par la table 1. Ici, $\delta = 8$. On suppose que le système ne peut charger que deux véhicules en même temps, i.e. $\widetilde{m} = 2$. Les véhicules $A$ et $E$ sont sur $c_0$, $C$ et $F$ sur $c_1$ et $B$ et $D$ sur $c_2$. Les parties grises correspondent aux instants où le véhicule est effectivement en charge.

Soit une première relation d'ordre $\leq^\star$ sur les demandes définie par $v \leq^\star u$ si l'immatriculation du véhicule associé à $v$ est inférieure ou égale au sens lexicographique à celle de $u$ (pour rappel, l'ordre lexicographique est l'ordre du dictionnaire de mots). On notera $v <^\star u$ si $v$ est différent de $u$ et $v$ est situé avant $u$ selon l'ordre lexicographique.

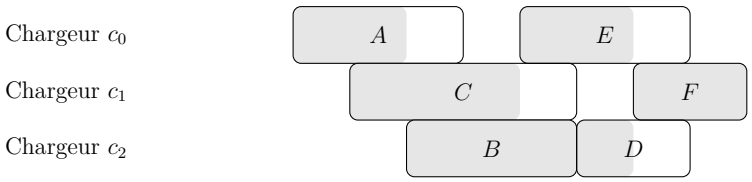



FIGURE 20 – Une allocation réalisable de la charge des véhicules pour l'exemple de la table 1 pour $\delta = 8$ et $\widetilde{m} = 2$. Les zones grisées correspondent aux instants effectifs de charge.

Soit également une seconde relation d'ordre $\preceq$ définie sur l'ensemble des couples $(v,t) \in \mathcal{D} \times \{0,\ldots,\delta\}$ par $(v_1,t_1) \preceq (v_2,t_2)$ si $v_1 <^\star v_2$ ou si $v_1 = v_2$ et $t_1 \leq t_2$. Par exemple, $(A,0) \preceq (A,1)$ et $(B,4) \preceq (C,1)$.

Pour limiter la taille de stockage de la matrice $x_{\star,\star}$, on stocke uniquement la liste `Liste_Xvt` des valeurs $(v,t)$ telles que $x_{v,t} = 1$ ordonnée par la relation d'orde $\preceq$ en ordre croissant (sans doublon).

(a) Donner la liste `Liste_Xvt` pour l'exemple représenté par la figure 20. Les cases grisées correspondent aux instants effectifs de charge ; pour simplifier la réponse, les éléments de $\mathcal{D}$ sont identifiés par la lettre associée ;

(b) donner en deux lignes maximum le principe d'un algorithme de complexité strictement meilleure que $\mathcal{O}(p)$ pour chercher si un couple $(v,t)$ est dans `Liste_Xvt` croissante. Ici, $p$ représente le nombre d'éléments de `Liste_Xvt`.

(c) démontrer la complexité de cet algorithme. Pour cela :

1. exprimer la suite $\alpha_p$ qui correspond au nombre maximum de comparaisons pour retrouver le couple dans une liste de $p$ éléments en fonction de $\alpha_{p-1},\ldots,\alpha_0$. Justifier votre réponse ;
2. résoudre la suite $\alpha_p$ pour $p = 2^k$ ;
3. en déduire pour tout entier $q > 0$ l'ordre de grandeur de $u_q$ et la complexité de l'algorithme dans le pire des cas. Pour cela, on supposera que la suite $\alpha_p$ est croissante.

**Question 40 :** Dans cette question, exprimer dans le cas général les contraintes et l'objectif du problème en utilisant les variables mathématiques $x_{v,t} \in \{0,1\}$.

(a) A tout instant $t \in \{0,\ldots,\delta-1\}$, il y a au plus $\widetilde{m}$ véhicules en charge ;

(b) le véhicule associé à la demande $v$ ne peut être effectivement chargé que dans l'intervalle $\{\overline{r}_v,\ldots,\underline{d}_v\}$ ;

(c) le véhicule associé à la demande $v$ est chargé pendant au plus $p_v$ intervalles ;

(d) il faut maximiser le nombre total de périodes effectives de charge.

**Question 41 :** On représente une solution réalisable associé à $x_{v,t}$, pour $(v,t) \in \mathcal{D} \times \{0,\ldots,\delta-1\}$ par un graphe orienté bipartie $X = (V_X, A_X)$ défini par :

— $V_X = \mathcal{D} \cup \{0,\ldots,\delta-1\}$ ;

— Pour tout couple $(v,t) \in \mathcal{D} \times \{0,\ldots,\delta-1\}$ on a deux cas :

1. Si $x_{v,t} = 1$, alors l'arc $(v,t) \in A_X$ ;



**Tournez la page S.V.P.**

2. sinon, l'arc $(t, v) \in A_X$.

Un graphe est bipartie si il est possible de partitionner des sommets en deux sous-ensembles disjoints tel que tout arc $a = (i, j)$ relie deux sommets dans des sous-ensembles différents. Dans le cas présent, ces deux sous-ensembles sont $\mathcal{D}$ et $\{0, \ldots, \delta\}$.

Pour tout sommet $s \in V_X$, l'ensemble $\Gamma_X^+(s)$ des successeurs de $s$ dans $X$ est defini par $\Gamma_X^+(s) = \{s' \in V_X, (s, s') \in A_X\}$. De même, l'ensemble $\Gamma_X^-(s)$ des précédecesseurs de $s$ dans $X$ est defini par $\Gamma_X^-(s) = \{s' \in V_X, (s', s) \in A_X\}$.

(a) Donner le graphe $X$ associé à la solution réalisable exprimée par la figure 20 ;

(b) pour tout sommet $s \in V_X$, donner les ensembles $\Gamma_X^+(s)$ et $\Gamma_X^-(s)$ pour le graphe $X$ associé à la figure 20.

**Question 42 :** Dans le cas général, on suppose que l'on a un graphe orienté bipartie quelconque $Y = (V_Y, A_Y)$ dont les deux sous-ensembles de sommets de la partition sont $\mathcal{D}$ et $\{0, \ldots, \delta - 1\}$. Soit également une valeur $\widetilde{m} > 0$.

(a) Donner dans le cas général l'ensemble des conditions qui permettent de s'assurer que $Y$ est associé à une solution réalisable pour le problème d'affectation des instants de charges pour au plus $\widetilde{m}$ chargeurs qui chargent à chaque instant. Justifier votre réponse ;

(b) exprimer le critère considéré en fonction de la cardinalité des ensembles de successeurs et/ou prédécesseurs. Justifier votre réponse ;

(c) quelle est la valeur du critère pour le graphe bipartie $X$ obtenu dans la question 41(a) et associé à la figure 20 ? Justifier votre réponse.

On souhaite développer un algorithme qui permette d'améliorer, si cela est possible, une allocation des temps de charges aux chargeurs.

Pour un graphe $X$ fixé, une demande $v$ est incomplète si $|\Gamma_X^+(v)| < p_v$. De même, un instant $t \in \{0, \ldots, \delta - 1\}$ est incomplet si $|\Gamma_X^-(t)| < \widetilde{m}$.

De plus, un chemin améliorant de $X$ est un chemin sans circuit dont le premier sommet est un élément $t \in \{0, \ldots, \delta - 1\}$ incomplet, et le dernier un élément $v \in \mathcal{D}$ également incomplet.

**Question 43 :** Dans cette question, on considère le graphe $X$ obtenu dans la question 41(a) et associé à la figure 20.

(a) Donner, si il en existe, les sommets incomplets pour le graphe bipartie $X$ ;

(b) donner un chemin améliorant $\mu$ de $X$. Ce chemin est-il unique ?

**Question 44 :** On suppose que le graphe $X$ possède un chemin améliorant $\mu$ de sommet de départ $t^\star$ et d'arrivée $v^\star$. On construit alors un autre graphe bipartie $X'$ en remplaçant tout arc $(x, y)$ du chemin $\mu$, par un arc $(y, x)$.

(a) Construire le graphe $X'$ à partir de celui obtenu dans la question 41(a) et associé à la figure 20. Donner la liste `Liste_Xvt_prime` associée à une nouvelle matrice $x'_{\star,\star}$ de $X'$. Est-ce que $x'_{\star,\star}$ est une solution réalisable du problème ? Est-elle optimale ?

(b) on se place maintenant dans le cas général. Soit alors $X$ un graphe bipartie associé à une solution réalisable $x_{\star,\star}$, $\mu$ un chemin améliorant et $X'$ le graphe construit

à partir de $\mu$ et $X$. Démontrer que la solution $x'_{\star,\star}$ est réalisable et strictement meilleure que $x_{\star,\star}$ ;

(c) Dans le cas général, est-ce que la solution $x'_{\star,\star}$ obtenue est toujours optimale ? Justifiez votre réponse.

## 4.2 Accès à la station de recharge

Pour éviter que l'ensemble des demandes V associées à des véhicules électriques ne dépasse l'ensemble de chargeurs C à un instant donné, il est nécessaire de contrôler l'accès à la station de recharge. Chaque fois que des demandes V sont inférieur aux chargeurs C, il n'y a pas de blocage ; mais s'elles sont supérieur à C, certains devront bloquer. Pour mettre en œuvre ce contrôle, nous avons besoin des processus de synchronisations liés à l'entrée et à la sortie de la station de recharge.

**Question 45 :** La synchronisation nécessaire peut être obtenue grâce aux variables de verrouillage et des instructions atomiques, comme `test_and_set`. L'instruction `test_and_set` permettent à la séquence de consultation et de modification de la variable de verrouillage d'être indivisible (instruction atomique). Quand elle est appelée, elle remplace atomiquement la valeur mémorisée par la nouvelle valeur donnée en argument, et renvoie la valeur mémorisée précédente.

```
typedef volatile long lock_t;

lock_t test_and_set (lock_t* lock) {
    (a)
}

void acquire(lock_t* lock){
    (b)
}

void release(lock_t* lock){
    (c)
}
```

(a) Remplir la fonction `test_and_set` en langage C.

(b) Remplir la fonction `acquire` qui permet d'acquérir le verrou en langage C.

(c) Remplir la fonction `release` qui permet de liberer le verrou en langage C.

(d) Lorsque le `test_and_set` est appelée en tant que fonction C, renvoie-t-elle 0 ou 1 lorsque vous avez réussi à acquérir le verrou ? Quelle est la valeur du verrou après ?

**Question 46 :** Cette question porte sur la mise en œuvre des deux processus de synchronisations liés à l'entrée et à la sortie de la station de recharge. En utilisant les fonctions de la question precedents `test_and_set`. Le processus d'entrée et le processus de sortie partagent une variable de verrouillage (`access`), qui est unique et est initialisée à 0, et



**Tournez la page S.V.P.**

une variable (`available_chargers`), qui donne le nombre de chargeurs disponibles une instance donnée et est initialisée à C.

```
lock_t access = 0;

void entry(){
(a)
}

void exit(){
(b)

}
```

(a) Remplissez la fonction d'entree (`entry`) dans la station de recharge afin d'éviter toute situation de concurrence.

(b) Remplissez la fonction de sortie (`exit`) dans la station de recharge afin d'éviter toute situation de concurrence.

Annexe

# A Schéma d'une borne

La fig21 présente un extrait d'une documentation d'une borne de recharge. La fig23 présente la courbe déterminant l'évolution du le courant maximum de recharge en fonction du rapport cyclique. La figure 22 présente le fonctionnement d'une borne en fonction du rapport cyclique.

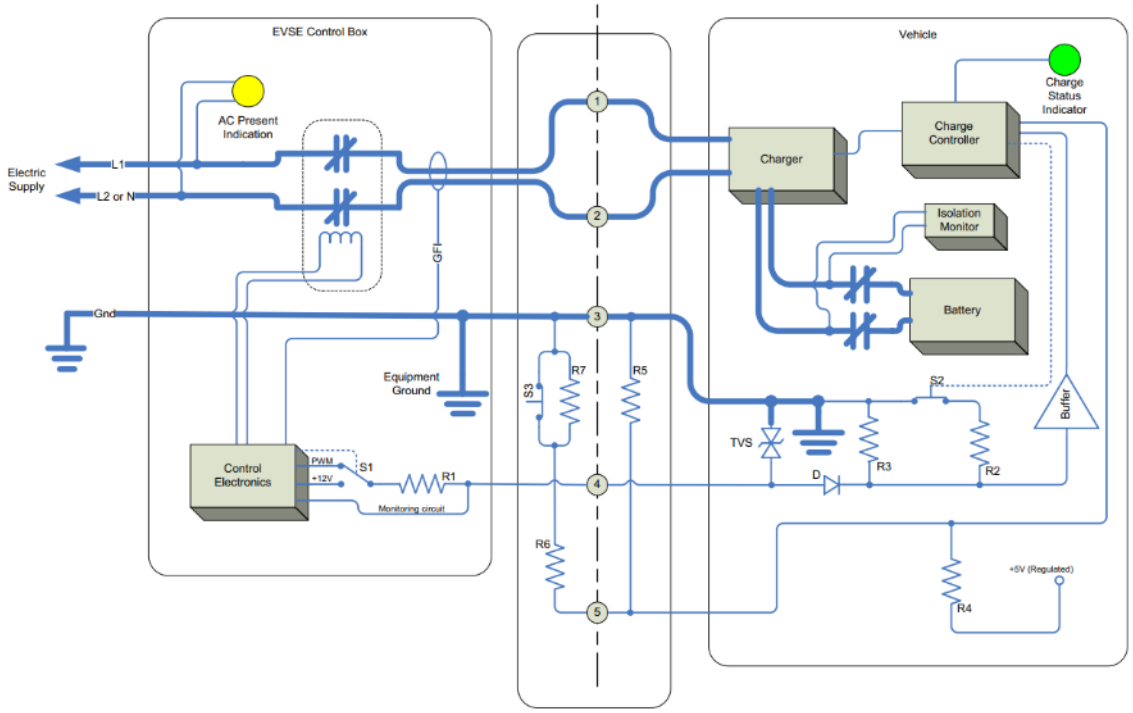



FIGURE 21 – Schéma de principe d'une borne de recharge.

| EVSE Nominal Duty Cycle | EVSE Commanded Maximum Current |
|---|---|
| Duty Cycle < 5% | Error state, no charging allowed |
| Duty Cycle = 5% | Indicates that digital communication is needed. |
| 5% < Duty Cycle < 10% | Error state, no charging allowed |
| 10% ≤ Duty Cycle ≤ 85% | Available current = (duty cycle %) x 0.6 |
| 85% < Duty Cycle ≤ 96% | Available current = (duty cycle % - 64) x 2.5 |
| Duty Cycle > 96% | Error state, no charging allowed |

FIGURE 22 – Fonctionnement borne en fonction du rapport cyclique



**Tournez la page S.V.P.**

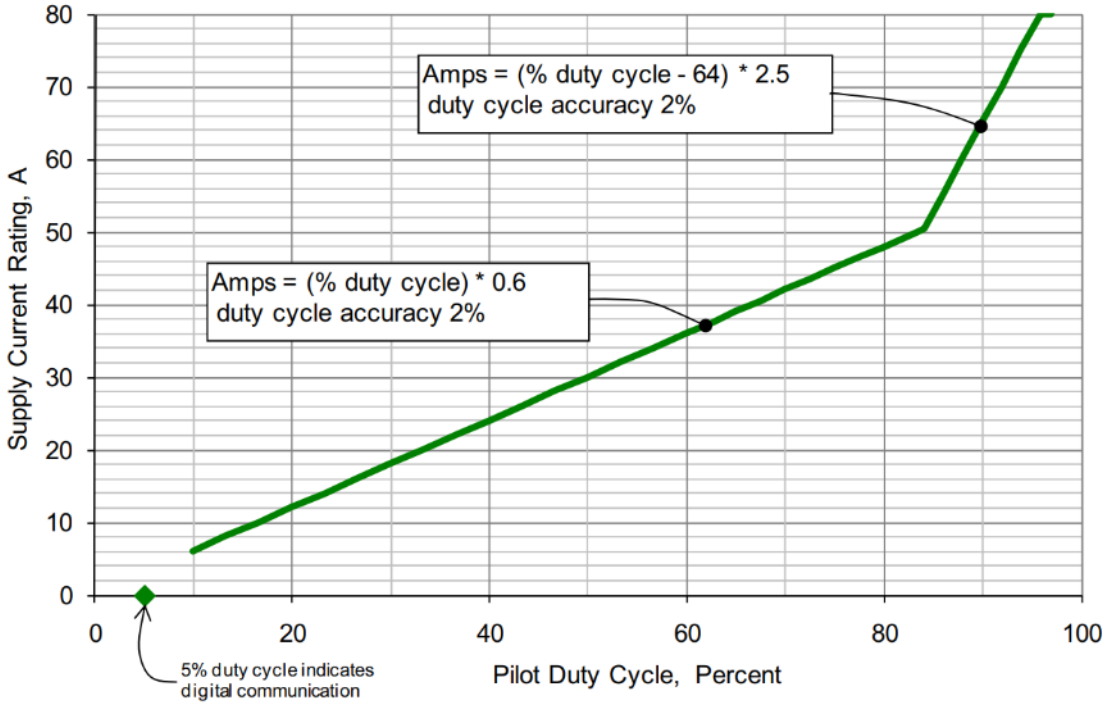



FIGURE 23 – Courant de charge max en fonction du rapport cyclique

# B Protocole Modbus

sources : https://www.virtual-serial-port.org/fr/articles/modbus-rtu-guide/

Le protocole Modbus RTU est un moyen de communication permettant l'échange de données entre des contrôleurs logiques programmables (PLC) et des ordinateurs. Les appareils électroniques peuvent échanger des informations via des lignes série à l'aide du protocole Modbus.

Ce protocole est très bien supporté et très utilisé dans la gestion technique de bâtiment (BMS) ainsi que les systèmes industriels automatisés (IAS). Son succès est dû à sa simplicité d'utilisation, à sa fiabilité et au fait qu'il soit open source, pouvant être utilisé pour tous types de périphériques et d'applications sans avoir à s'acquitter de droits d'auteur.

Le protocole a été développé et publié par Modicon® en 1979 pour être utilisé avec ses contrôleurs logiques programmables. Il est basé sur une architecture maître/esclave et supporte les périphériques série utilisant les protocoles RS232/RS485/RS422. Modbus intervient souvent dans les cas où de nombreux appareils de mesure et de contrôle doivent transmettre des signaux vers un système ou un contrôleur central pour récupérer des données et les analyser. L'automatisation industrielle et les systèmes de contrôle et d'acquisition de données en temps réel (SCADA) utilisent également le protocole Modbus de manière intensive.

## B.1 Que signifie Modbus RTU ?

Modbus RTU (Remote Terminal Unit) est l'un des deux modes de transmission définis dans la spécification Modbus originale. Ces deux modes, Modbus RTU et ASCII, sont conçus pour être utilisés avec les périphériques série supportant les protocoles RS232, RS485 et RS422. L'une des caractéristiques spécifiques de Modbus RTU est son utilisation du codage binaire et sa méthode de recherche d'erreurs CRC avancée. Modbus RTU est l'implémentation du protocole Modbus la plus utilisée pour les applications industrielles et les systèmes de production automatisés.

## B.2 Qu'est-ce qu'un maître Modbus RTU ?

Le maître Modbus RTU est le périphérique central demandant des informations aux périphériques esclaves qui y sont connectés. Un contrôleur central dans un système de production automatisé peut jouer le rôle d'un maître Modbus RTU. Une implémentation Modbus ne peut avoir qu'un seul maître. Les périphériques maîtres collectent les informations des esclaves et peuvent également écrire dans les registres des périphériques esclaves.

## B.3 Qu'est-ce qu'un esclave Modbus RTU ?

L'esclave Modbus RTU est le périphérique répondant aux requêtes envoyées par le maître. Il ne peut pas initier les transferts d'informations et est en phase d'attente tant qu'il n'a pas à répondre à une requête du maître.

Comme nous l'avons indiqué, il y a un seul périphérique maître dans une implémentation Modbus RTU et il peut y avoir jusqu'à 247 périphériques esclaves. Chaque esclave est identifié par une adresse numérotée de 1 à 247.

Au cœur du protocole Modbus se trouve le composant appelé Protocol Data Unit (PDU). Le PDU se compose de données et d'un code de fonction créés toujours de la même manière quel que soit le mode de transmission Modbus utilisé. Le code de fonction indique quelles données sont demandées par le maître.

Dans le mode de transmission Modbus RTU, des informations supplémentaires sont ajoutées autour du PDU pour former l'Application Data Unit (ADU). En mode Modbus RTU, un identifiant d'esclave de 1 octet est envoyé dans le flux du signal, avant le code de fonction, pour indiquer le périphérique esclave devant répondre à la requête. Un CRC de 2 octets est également ajouté au PDU pour s'assurer que la bonne quantité de données a été envoyée et reçue.

Les périphériques Modbus supportent quatre tableaux de données qui sont utilisés pour faciliter la communication entre eux. Il s'agit des entrées distinctes, sorties distinctes (bobines), registres d'entrée et registres d'attente. Les registres effectuent différentes fonctions et ne sont pas tous inclus dans chaque périphérique. Dans certains cas, seuls les registres d'attente sont utilisés pour les fonctionnalités d'entrée/sortie. Les codes de fonction indiquent comment le maître interagit avec le périphérique esclave spécifié par l'identifiant de ce dernier. Selon le code de fonction envoyé, le périphérique maître peut lire l'un des registres de



**Tournez la page S.V.P.**

| Domaine | Accès | Taille | Description |
|---|---|---|---|
| Entrées distinctes | lecture seule | 1 bit | utilisé comme entrée |
| Sorties bobines | lecture/ écriture | 1 bit | utilisé pour contrôler la sortie distincte |
| Registres d'entrée | lecture seule | 16 bits | utilisés pour l'entrée |
| Registres d'attente | lecture/ écriture | 16 bits | utilisés pour différents éléments tels que les entrées, les sorties, les données de configuration, etc. |

FIGURE 24 – Application Modbus

l'esclave ou y écrire.

Les esclaves renvoient des codes d'erreur lorsqu'ils reçoivent un paquet contenant une erreur de requête. Des codes d'erreur sont envoyés en cas de problèmes tels que la demande d'une fonction illégale, des adresses de registre illégales ne pouvant être contactées par l'esclave indiqué, et des messages indiquant que le périphérique esclave est occupé ou a rencontré un problème.

Modbus RTU vous oblige à connaître ou à définir des paramètres tels que la rapidité de modulation, le format des caractères (8 bits sans parité, etc) et l'identifiant de l'esclave en établissant une communication. Une erreur dans l'un de ces paramètres empêchera toute communication.

## B.4 Format de la trame Modbus

| Adresse | Fonction | Donnees | CRC |
|---|---|---|---|

FIGURE 25 – Trame Modbus

## B.5 Exemple d'un dialogue entre un Maître et un esclave

Tableau des différentes fonctions possibles en Modbus. fig.26

Pour lire une grandeur codée sur 16 bits s'adressant au mesureur, la trame à fournir se trouve en fig.27

Réponse du mesureur en fig ;28

Evolution dans le temps des données sur le bus Modbus fig.29

| Fonctions | |
|---|---|
| F1 | Lecture de n bits de sorties consécutifs |
| F2 | Lecture de n bits d'entrées consécutifs |
| F3 | Lecture de n mots de sortie consécutifs |
| F4 | Lecture de n mots d'entrées consécutifs |
| F5 | Ecriture de 1 bit |
| F6 | Ecriture de 1 mot |
| F7 | Lecture rapide de 8 bits |
| F8 | Diagnostic des échanges |
| F11 | Lecture du compteur d'événement |
| F12 | Lecture du buffer trace |
| F13 | Lecture/Ecriture adresse de statut d'exception |
| F14 | Diagnostique |
| F15 | Ecriture de n bits |
| F16 | Ecriture de n mots |

FIGURE 26 – Fonction Modbus

| Adresse | Fonction | Donnees | CRC |
|---|---|---|---|
| 0x05 | 0x03 | 0x01 0x31 0x00 0x01 | 0xD5BD |

FIGURE 27 – Trame Modbus Maître

| Adresse | Fonction | Donnees | CRC |
|---|---|---|---|
| 0x05 | 0x03 | 0x02 0x54 0x04 | 0x7687 |

FIGURE 28 – Trame Modbus Esclave

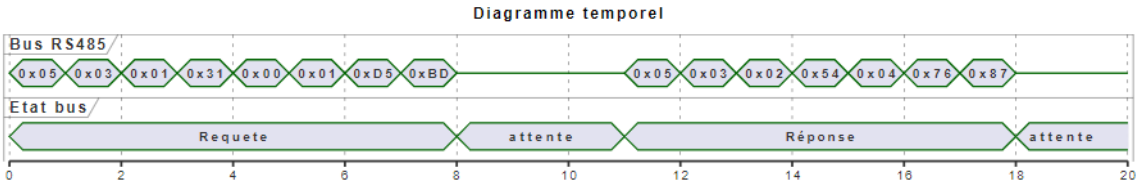



FIGURE 29 – Trame Modbus temporelle

## B.6 Module Orno WE514

Le module Orno WE514 est un dispositif réalisant des mesures d'énergie, d'intensité, de puissance, de tension.. Il s'agit d'un compteur d'énergie électrique communiquant en Modbus.

Trame Modbus fig.25

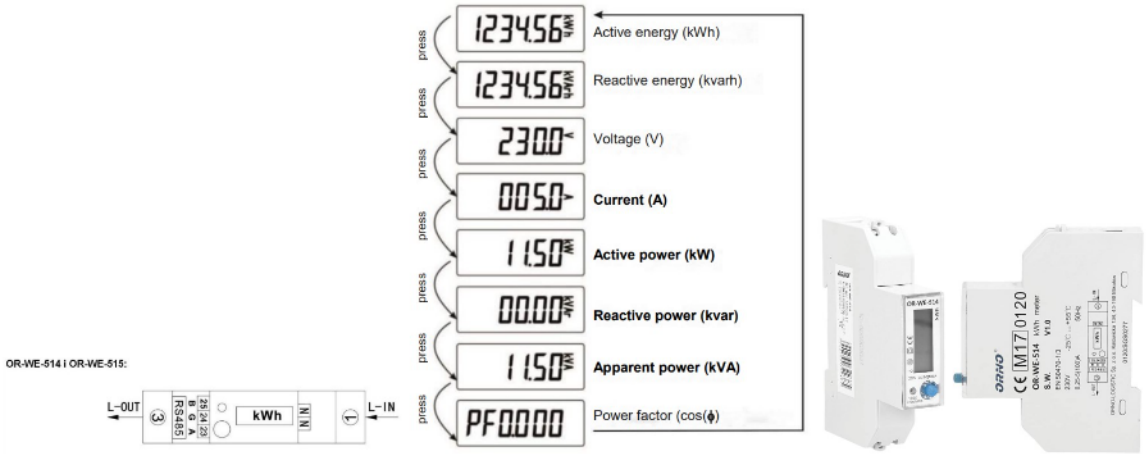
La donnée comporte l'adresse(dataAdress) du registre et le nombre de registre(register). Un registre correspond a une donnée codée sur 16 bits.fig.32



**Tournez la page S.V.P.**

Orno WE514

| | |
|---|---|
| OR-WE-512 | Single-phase meter 100A |
| OR-WE-514 | Single-phase meter 100A with port RS-485 |
| OR-WE-515 | Single-phase multitariff meter 100A with port RS-485 |
| ORNO-LOGISTIC Sp. z o.o.<br>ul. Katowicka 134<br>43-190 Mikołów<br>tel. 32 43 43 110 | Service and assembly manual |




Extrait de la documentation technique :

FIGURE 30 – Documentation OrnoWE514

| Adresse | Fonction | Donnees | CRC |
|---|---|---|---|

FIGURE 31 – Format trame Modbus

| Adresse | Fonction | dataAdress | register | CRC |
|---|---|---|---|---|
| 0x05 | 0x03 | 0x01 0x31 | 0x00 0x01 | 0xD5BD |

FIGURE 32 – Format de la trame pour une requete

Réponse du module Orno fig.33

| Adresse | Fonction | Nbr d'octets | Valeur | CRC |
|---|---|---|---|---|
| 1byte | 1byte | 1byte | dépend nbr octets | 2byte |

FIGURE 33 – Réponse du module Orno

| ORNO | | |
|---|---|---|
| type | Mesureur | |
| ref | Orno514 | |
| frequency | register | 1 |
| | dataAdress | 130H |
| | unit | 0.01 |
| | mode | read |
| voltageV1 | register | 1 |
| | dataAdress | 131H |
| | unit | 0.01 |
| | mode | read |
| currentV1 | register | 2 |
| | dataAdress | 139H |
| | unit | 0.001 |
| | mode | read |
| powerActiveV1 | register | 2 |
| | dataAdress | 140H |
| | unit | 0.001 |
| | mode | read |
| sumPowerActiveV1 | register | 2 |
| | dataAdress | 146H |
| | unit | 0.001 |
| | mode | read |
| powerReactiveV1 | register | 2 |
| | dataAdress | 148H |
| | unit | 0.001 |
| | mode | read |
| powerApparentV1 | register | 2 |
| | dataAdress | 150H |
| | unit | 0.001 |
| | mode | read |
| sumPowerReactiveV11 | register | 2 |
| | dataAdress | 14EH |
| | unit | 0.01 |
| | mode | read |
| energyActvie | register | 5 |
| | dataAdress | A000H |
| | unit | 0.001 |
| | mode | read |

| | | | |
|---|---|---|---|
| config | adress | adress | 5 |
| | | register | 1 |
| | | dataAdress | 110H |
| | | mode | read-write |
| | bauds | bauds | 4 |
| | | register | 1 |
| | | dataAdress | 111H |
| | | mode | read-write |
| | parity | parity | even |
| | | register | 1 |
| | | dataAdress | x |
| | | mode | read-write |
| | stop | stop | 1 |
| | | register | 1 |
| | | dataAdress | x |
| | | mode | read-write |
| | format | format | 8 |
| | | register | 1 |
| | | dataAdress | x |
| | | mode | read-write |

FIGURE 34 – Registres internes et configuration

Registre interne et Configuration du module Orno WE514 fig.34

# C Contacteur de puissance

— Contacteur de puissance fig.35.
— Relais Modbus commandant le contacteur de puissance fig.36.
— Description sur le fonctionnement du relais fig.37.

# D Organisation des fichiers

L'organisation des fichiers se trouve fig.38



**Tournez la page S.V.P.**

| | |
|---|---|
| Le contacteur est contitué d'un relais faible puissance qui commande un contacteur de plus forte puissance. |  |

FIGURE 35 – Contacteur de puissance

| Format | Description | Image |
|---|---|---|
| Baud rate :9600 8 NONE 1 | Relais de faible puissance commandé en Modbus RTU |  |

FIGURE 36 – Relais commandant le contacteur

# E Fichiers

— Fichier Borne.js en fig ;39
— Fichier Contacteur.js en fig ;40
— Fichier Mesureur.js en fig ;41
— Fichier Gestion.js en fig ;39
— Fichier Programme principal en fig ;43
— Fichier eventPortSerie.js en fig ;44

1. Interface de communication RS485
2. Interface de communication TTL
3. Entrée d'isolement de l'optocoupleur 2 canaux
4. Sortie d'isolation de optocoupleur à 2 canaux
5. Un bouton d'utilisateur
6. Un indicateur utilisateur LED
7. Un indicateur de puissance
8. Un microcontrôleur STM32F030F4
9. 2 voyants d'état de relais LED
10. Interface du terminal d'alimentation (alimentation 12V)

**Set the address to: 09**

- Trame : 01 10 00 00 00 01 02 00 09 66 56

//The current address 01 is modified to 09

- Trame : 00 10 00 00 00 01 02 00 09 6B C6

//The broadcast address is changed to 09

**Fonctionnement**

[Address 1]

- Relay No. 0 is on: 01 05 00 00 FF 00 8C 3A
- Relay No. 0 is closed: 01 05 00 00 00 00 CD CA
- Relay No. 1 is open: 01 05 00 01 FF 00 DD FA
- Relay No. 1 is closed: 01 05 00 01 00 00 9C 0A

Pour connaitre l'état des relais

- Trame : 01 01 00 00 00 08 3D CC Interrogation à l'adresse 01.
- Réponse : 01 01 01 02 D0 49

La donnée est 02 qui correspond au Relais 1 On Relais0 Off

FIGURE 37 – Fonctionnement des relais commandés en Modbus



**Tournez la page S.V.P.**

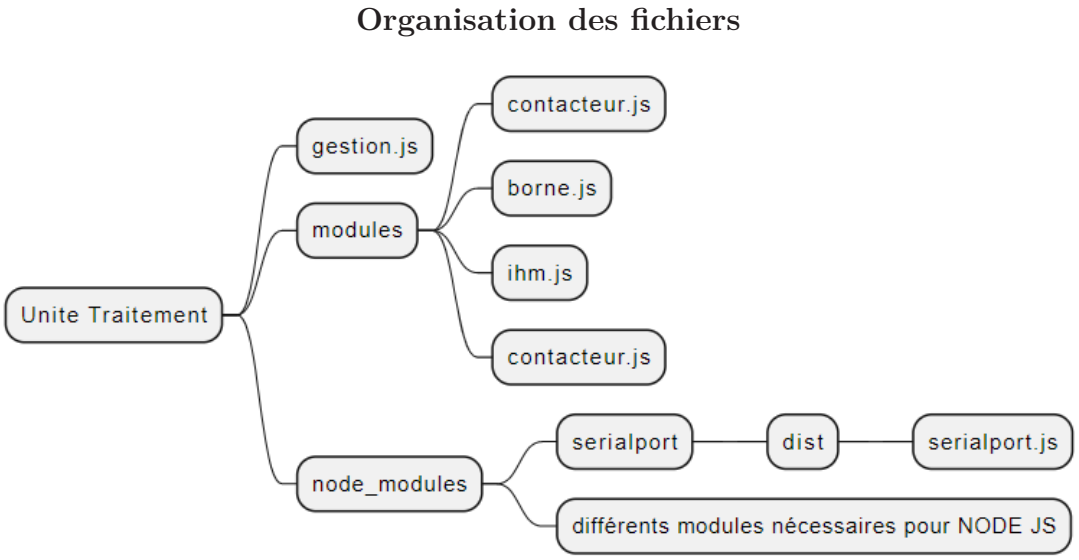



FIGURE 38 – Organisation des différentes sources dans l'espace de travail

**Fichier Borne.js**

```
Classe Borne.js
import { Contacteur } from "./contacteur.js"
import {IHM} from "./ihm.js"
import { Mesureur } from "./mesureur.js"
class Borne{
    constructor(adresseMesureur=10,adresseContacteur=20,adresseIHM=30){
      this._Mesureur=new Mesureur(adresseMesureur)
      this._Contacteur=new Contacteur(adresseContacteur)
      this._IHM=new IHM(adresseIHM)
    }
    actif(){
      return this.ajouterCRC(this._Contacteur.on())
    }
    inactif(){
      return this.ajouterCRC(this._Contacteur.off())
    }
    tension(){
      return this.ajouterCRC(this._Mesureur.voltageV1())
    }
    intensite(){
      return this.ajouterCRC(this._Mesureur.currentV1())
    }
    puissance(){
      return  this.ajouterCRC(this._Mesureur.powerActiveV1())
    }
    CRC(data) {
        ..
        return crc;
      }
    ajouterCRC(data){
      let trame=data
      let CRC=  this.CRC(trame)
      trame.push(Math.trunc(CRC%256))
      trame.push(Math.trunc(CRC/256))
      return trame
      }
}

export {Borne}
```

FIGURE 39 – Fichier borne.js



**Tournez la page S.V.P.**

**Fichier contacteur.js**

```
export {Contacteur}
class Contacteur{
    constructor(adresseRelais=2){
        this._adresseRelais=adresseRelais
        this._etat=false
    }
    on(){..}
    off(){..}
}
```

FIGURE 40 – Fichier contacteur.js

**Fichier mesureur.js**

```
export{Mesureur}
class Mesureur{
    constructor(adresseOrno=20){
        this._adresseOrno=adresseOrno
    }
    voltageV1(){..}
    currentV1(){..}
    powerActiveV1(){
        let trame=[]
        const fonction=0x03
        trame[0]=this._adresseOrno
        trame[1]=fonction
        trame[2]=0x01
        trame[3]=0x40
        trame[4]=0x00
        trame[5]=0x02
        return trame
        }
    frequency(){..}
    configAdress(adress){..}
    configBauds(bauds){..}
    configParity(parity){..}
}
```

FIGURE 41 – Fichier mesureur.js

**Fichier gestion.js**

```
import { Borne } from "./modules/borne.js"
import {SerialPort} from "serialport"

class Gestionnaire{
    constructor(){
        this.borne1=new Borne(0x11,0x21,0x31)
        this.borne2=new Borne(0x12,0x22,0x32)
        this.borne3=new Borne(0x13,0x23,0x33)
        this.borne4=new Borne(0x05,0x10,0x50)
        this.portCom="COM3"
        this.port=this.initSerie()
    }
    // Methodes pour le dialogue avec le port  Serie
    initSerie(){
        return new SerialPort(
        {
            path:this.portCom,
            baudRate:9600,
            parity:"none",
            dataBits:8,
            stopBits:1
        },
        (err)=>{
            if(err){
                console.log("erreur ",err.message)
            }
        })
    }
    write(data){..}
    read(){..}
    // Gestion des differentes bornes
    gestion(){
    }
}
```

FIGURE 42 – Fichier gestion.js



**Tournez la page S.V.P.**

**Programme principal**

```
// Programme Principal
const gestion=new Gestionnaire()
console.log("Production de Trames")
console.log("trame Relais On ",gestion.borne1.actif())
console.log("trame Lecture Tension ",gestion.borne2.tension())
console.log("trame Lecture Intensite",gestion.borne3.intensite())
console.log("trame Lecture Puissance",gestion.borne4.puissance())
```

FIGURE 43 – Programme principal

**Fichier eventPortSerie.js**

```
// Cette fonction realise bien une communication serie
//      -> Ouverture & Fermeture du port
//      -> Ecriture & Lecture de donnees


import { SerialPort } from "serialport";
// Dans le module SerialPort les emetteurs sont implementes
import { EventEmitter, errorMonitor } from "node:events";

export const eventPortSerie = new EventEmitter();

const portCom1 = "/dev/ttyUSB1";
const port1 = new SerialPort(
  {
    path: portCom1,
    baudRate: 9600,
    parity: "none",
    databits: 8,
    stopbits: 1,
  },
  (err) => {
    if (err) {
      console.log("erreur ", err.message);
    }
  }
);
// Mise en place des differents ecouteurs
port1.on("open", showPortOpen);      // A l'ouverture
port1.on("data", readSerialData);
// A la reception de donnee dans le buffer
port1.on("close", showPortClose);    // A la fermeture du port
port1.on("error", showError);        // Si erreur de transmission
```

FIGURE 44 – Fichier eventPortSerie.js



**Tournez la page S.V.P.**

**Suite Fichier eventPortSerie.js**

```
/*
Mise en place d'un ecouteur sur le canal "emettre" de l'objet eventPortSerie.
Ce canal permet de recevoir de l'exterieur puis d'envoyer des donnees
sur le port serie.
*/
eventPortSerie.on("emettre", envoieDonnee)
function envoieDonnee(data){
  write(data);
  // Fonction permettant d'envoyer des donnees sur le port COM
}
function showPortOpen() {
  const message = `Le port serie est connecte sur ${portCom1}
  avec un baudRate de ${port1.baudRate}`;
  console.log(message);
  eventPortSerie.emit("start", "le port serie est disponible");
}
function readSerialData(data) {
  eventPortSerie.emit("DataOk", data);
}
function showPortClose() {
  console.log("port closed.");
}
function showError(error) {
  console.log("Serial port error: " + error);
  eventEmitter.emit("Erreur","erreur Com");
}
function write(donnee) {
  if (port1.isOpen == true) {
    port1.write(donnee);
    eventPortSerie.emit("travail", "ok");
  } else {
    console.log("le port est ferme");
    eventPortSerie.emit("Erreur", "Port ferme");
  }
}
```

FIGURE 45 – Fichier eventPortSerie.js suite

Modèle CMEN v3

**Nom de famille :**
*(Suivi, s'il y a lieu, du nom d'usage)*

**Prénom(s) :**

**Numéro Candidat :**

**Né(e) le :** / /

*Cadre réservé aux candidats de concours de recrutement et examens professionnels*

**Concours :** ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

**Examen professionnel pour l'avancement au grade de :** .....

*Cadre réservé aux candidats d'examens et du concours général*

**Examen :** ..... **Série / Spécialité :** .....

**Epreuve - Matière :** ..... **Session :** .....

EAE SIN 3

# Documents réponses
# Questions 4, 7 et 13

## Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

# Document Réponses

## 1 Réponse à la question 4

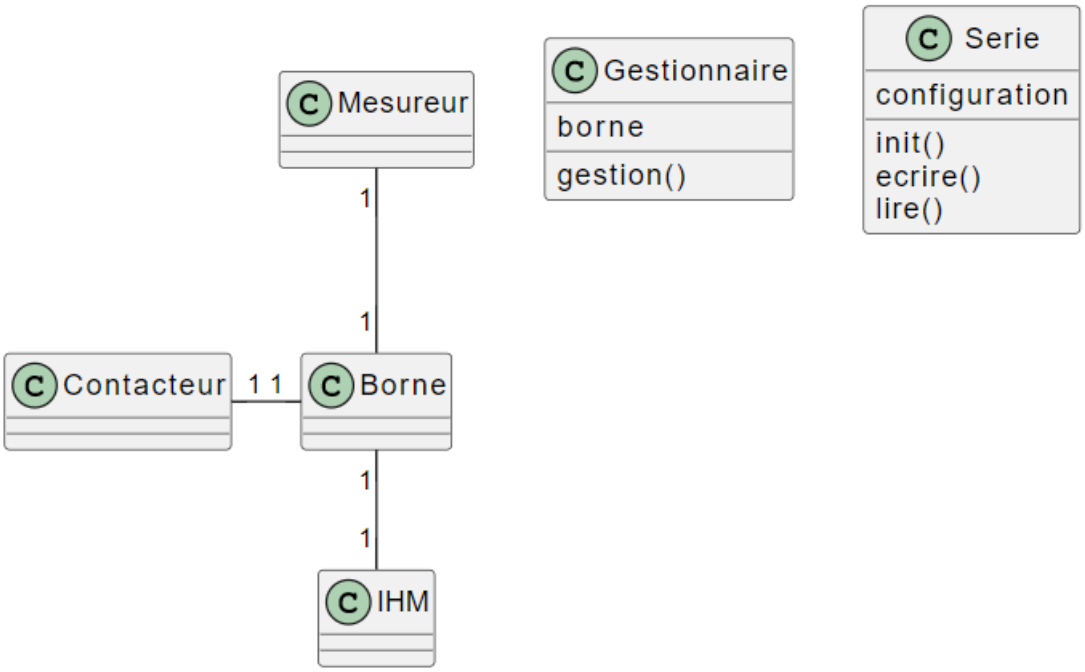
| ORNO | | |
|---|---|---|
| adresse | ? | |
| fonction | ? | |
| Donnee | ? | |
| CRC | ? | |
| Exploitation Requete | register | ? |
| | dataAdress | ? |
| | type Mesure | ? |

## 2 Réponse à la question 7

```
on(){
    let trame=[]
    this._etat=true
    const fonction=?
    trame[0]=this._adresseRelais
    trame[1]=fonction
    trame[2]=?
    trame[3]=?
    trame[4]=?
    trame[5]=?
    return trame
```

```
}
off(){
    let trame=[]
    this._etat=false
    const fonction=?
    trame[0]=this._adresseRelais
    trame[1]=fonction
    trame[2]=?
    trame[3]=?
    trame[4]=?
    trame[5]=?
    return trame
}
```

## 3 Réponse à la question 13



Modèle CMEN v3

**Nom de famille :**
*(Suivi, s'il y a lieu, du nom d'usage)*

**Prénom(s) :**

**Numéro Candidat :**

**Né(e) le :** / /

*Cadre réservé aux candidats de concours de recrutement et examens professionnels*

**Concours :** ..... **Option / Section :** ..... **N° d'inscription :**

Cocher une seule case parmi les six types de concours suivants :

☐ externe ☐ 3e externe ☐ externe spécial ☐ interne ou 1er interne ☐ 2nd interne ☐ 2nd interne spécial

Cocher public OU privé UNIQUEMENT pour les concours enseignants : ☐ public ☐ privé

**Examen professionnel pour l'avancement au grade de :** .....

*Cadre réservé aux candidats d'examens et du concours général*

**Examen :** ..... **Série / Spécialité :** .....

**Epreuve - Matière :** ..... **Session :** .....

EAE SIN 3

# Documents réponses
# Questions 23 et 27

## Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

## 4 Réponse à la question 23

| Etat en précédent | Action | Etat suivant |
|---|---|---|
| Arret | Travail à faire | Start |
| Start | Départ cycle | Emission |
| Emission | **Trame0** émise | Reception |
| Reception | Trame Recue? | si non Reception |
| Reception | Trame Recue? | si oui Trame suivante |
| Trame suivante | Fin de cycle? | si oui ARRET |
| Trame suivante | Fin de cycle? | si non Emission |
| Emission | **Trame1** émise | Reception |
| Reception | En attente | Reception |
| Reception | Time Out | |
| Trame suivante | Fin de cycle? | |
| Trame suivante | Fin de cycle? | |
| Emission | | Reception |
| Reception | Trame Recue? | si non Reception |
| Reception | Trame Recue? | si oui Trame suivante |
| Trame suivante | Fin de cycle? | |

## 5 Réponse à la question 27

Les écouteurs :

| Objet | Canal | Action | Role |
|---|---|---|---|
| port1 | open | showPortOpen | ? |
| port1 | data | ? | ? |
| port1 | close | showPortClose | Permet d'afficher sur la console l'état du port série "Fermé" |
| port1 | error | showError | Permet le traitement d'erreur |
| eventPortSerie | ? | ? | ? |

Les émetteurs

| Objet | Canal | Données | Role |
|---|---|---|---|
| eventPortSerie | start | "Le port série est disponible" | ? |
| eventPortSerie | ? | ? | ? |
| eventPortSerie | Erreur | "erreur Com" | ? |
| eventPortSerie | ? | ? | ? |
| eventPortSerie | Erreur | "Port fermé" | Indique que le port série est fermé |