Dossier De Conception (DDC)

du projet

Kart à Hélice

Responsabilité documentaire

Action	NOM Prénom	Fonction	Date	Signature
Rédigé par	Paul Pelamatti Nathan Petitjean Adrien Dumas Amory Cazade Sullivan Hortos Quentin Bernyer Gueroc Mantaux	Technicien	09/02/2022	
Approuvé par	M.Moutault M.Tartaglione (IUT GEII Bdx)	Chef de projet	09/02/2022	
Approuvé par	M.Moutault M.Tartaglione	Client	09/02/2022	

Suivi des révisions documentaires

Indice	Date	Nature de la révision
1	01/09/2021	Publication préliminaire du DDC, document à compléter par le Technicien
2	09/02/2022	Première publication

Documents de références

Sigle	Référence	Titre	Rév.	Origine
[CDC]	KAH_CDC	Cahier des charges	1	M.Moutault M.Tartaglione

Table des matières

I-Nature du document

Ce document est un dossier de conception et a pour but de détailler la conception du produit développé. Il apporte ainsi des preuves de la conformité du produit par rapport à l'ensemble des exigences client. Le paragraphe 3 du CDC décrit de façon plus détaillée la nature et le positionnement de ce document dans l'arborescence documentaire du projet.

II-Conception préliminaire du produit

Ce chapitre décrit l'architecture fonctionnelle du produit. Il apporte les premiers éléments de preuve de la faisabilité du produit vis-à-vis des exigences client.

1 Architecture Mécanique

1.1 Architecture Mécanique Émetteur

Référence de préconception: PRC01

Exigences client vérifiées par préconception : EXIG_EMTT_DIMENSIONS, EXIG_EMTT_LOGO, EXIG_EMTT_INTERRUPTEUR, EXIG_EMTT_IHM, EXIG_EMTT_KLAXON, EXIG_EMTT_INDICATEUR

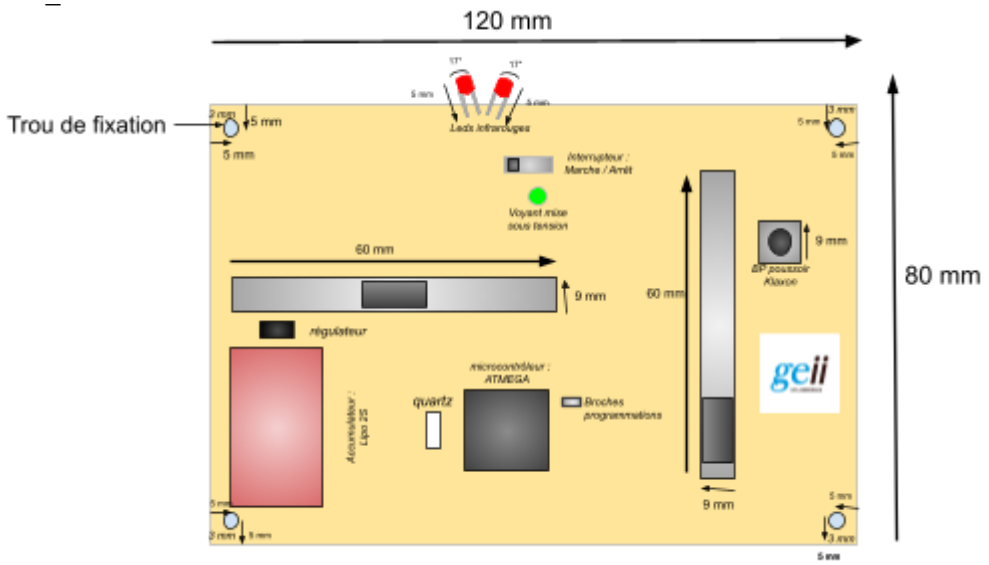


Figure 1 : Architecture mécanique carte émettrice

Détails de la carte émettrice :

- Le logo GEII est centré à droite pour une meilleur esthétique
- Le potentiomètre rectiligne de vitesse est placé verticalement pour faciliter l'augmentation/diminution la vitesse du kart
- Le potentiomètre rectiligne de direction est placé horizontalement pour favoriser le changement d'état droite/gauche du kart.
- Le bouton poussoir du klaxon est placé près du potentiomètre rectiligne de vitesse pour avoir une meilleure accessibilité au niveau de la commande.
- La LED indicatrice verte est placée en dessous de l'interrupteur pour voir si la mise sous tension est effective.
- L'ATMEGA328P est centré au milieu pour laisser la place et mieux répartir les autres composants.

1.2 Architecture Mécanique Récepteur

Référence de préconception: PRC02

Exigences client vérifiées par préconception : EXIG_RCPT_DIMENSIONS, EXIG_RCPT_LOGO, EXIG_RCPT_INDICATEUR, EXIG_RCPT_CONNEXION, EXIG_RCPT_KLAXON

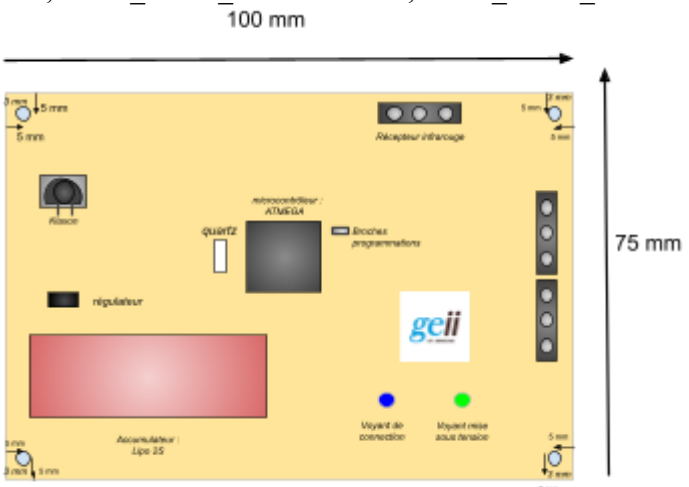


Figure 2 : Architecture mécanique carte réceptrice

Détails de la carte émettrice :

- Le logo GEII et le n° d'équipe est centré à droite pour une meilleure esthétique.
- Les 2 emplacements de 3 pin nécessaire au moteur sont situés sur partie droite de la carte
- Le klaxon (buzzer) est situé sur la partie gauche.
- Les LED bleu et verte sont placées en bas pour voir si la mise sous tension est effective et si la connexion est réalisée.
- L'ATMEGA328P est centré au milieu pour laisser la place et mieux répartir les autres composants.
-

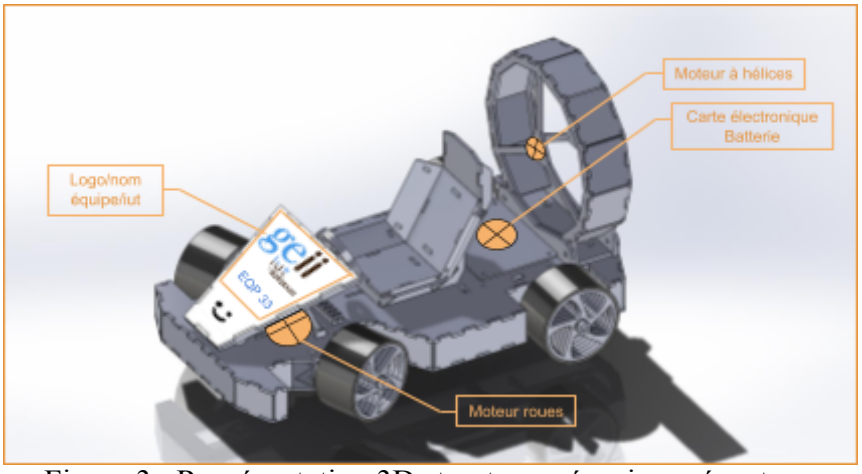


Figure 3 : Représentation 3D structure mécanique récepteur.

2 Architecture Électronique C1-03 C1-09 C1-10

2.1 Architecture Émetteur

Référence de préconception: PRC03

Exigences client vérifiées par préconception : : EXIG_EMTT_IHM, EXIG_EMTT_KLAXON, EXIG_EMTT_TRAITEMENT, EX IG_EMTT_REPETI TIVITE, EXIG_EMTT_RETENTISSEMENT, EXIG_EMTT_ENERGIE, EXIG_EMTT_INTERRUPTEUR, EXIG_EMTT_PUISSANCE, EXIG_EMTT_INDICATEUR **C1-22**

L'architecture du produit s'articule en 4 blocs

- Le bloc ACQUISITION D'INFORMATIONS comporte tous les composants d'interface machine.
- Le bloc TRAITEMENT D'INFORMATIONS comporte le coeur de traitement permettant de faire l'acquisition des données des interfaces homme machine (potentiomètre de vitesse et de direction ainsi que le bouton poussoir du klaxon), le traitement de ces données , la création d'une trame NEC et l'émission de celle-ci sur une de ses sorties.
- Le bloc ACTIONS comporte tous les composants actifs.
- Le bloc ÉNERGIE comporte les organes fournissant l'alimentation et la régulation de la carte.

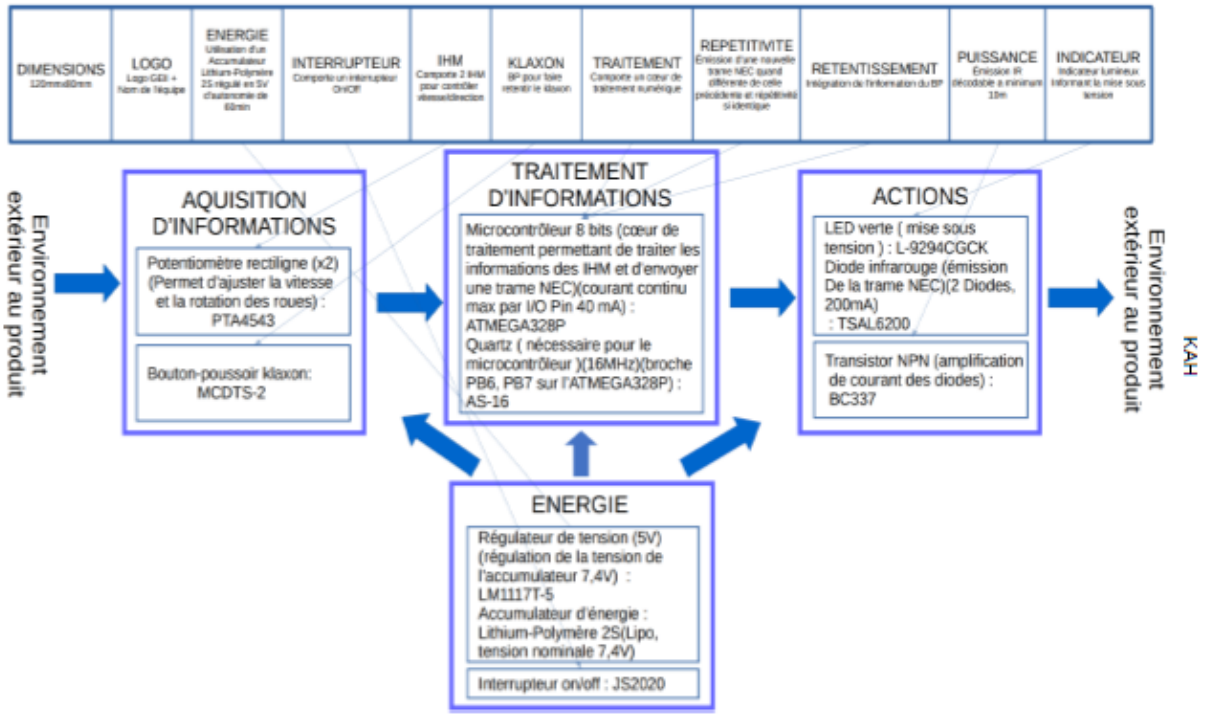


Figure 4 : Architecture électronique carte émetteur

2.1.1 Bloc Acquisition

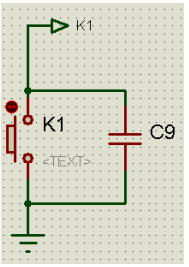
Référence de préconception: PRC04

Exigences client vérifiées par préconception : EXIG_EMTT_IHM, EXIG_EMTT_ KLAXON C1-22

D'après l'exigence EXIG_EMTT_IHM **:**

Conformément à l'exigence, l'émetteur comporte 2 potentiomètres placés entre la masse et le régulateur linéaire. Leur sorties sont reliées à des entrées analogiques du microcontrôleur.

D'après l'exigence EXIG_EMTT_KLAXON **:** L'émetteur comporte un bouton-poussoir sur lequel l'utilisateur peut appuyer pour indiquer qu'il souhaite faire retentir le klaxon du kartz. Il est relié à la masse puis à une broche numérique du microcontrôleur avec la résistance de pullup activé.



2.1.2 Bloc traitement

Référence de préconception: PRC05

Exigences client vérifiées par préconception : EXIG_EMTT_TRAITEMENT, EXIG_EMTT_REPETITIVITE, EXIG_EMTT_RETENTISSEMENT C1-22

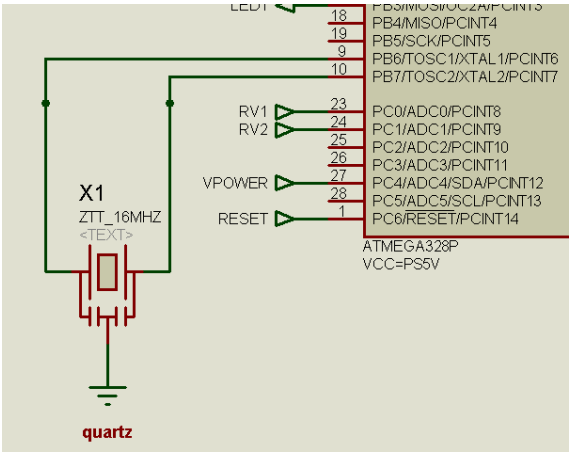
Il nous faut un coeur de traitement pour

- acquérir numériser encoder construire et émettre du code
- Nous choisissons le ATMEGA 328p car bien qu' il ne soit pas le plus puissant ni le plus rapide, il correspond aux exigences, de plus son prix et sa consommation sont peu élevés.

Référence de préconception: PRC06

Exigences client vérifiées par préconception :EXIG_EMTT_REPETITIVITE

- Le circuit électronique de la carte émettrice comporte un quartz pour aider le cœur de traitement à respecter les périodes entre les émissions des trames.
- Le quartz choisi est le AS-16 oscillant à 16 MHz, il est peu coûteux et facile d'utilisation.
- le quartz doit être connecter sur les broches XTAL 1 et 2



2.1.3 Bloc action

Référence de préconception: PRC07

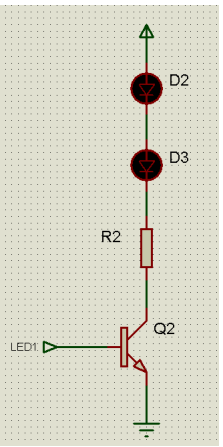
Exigences client vérifiées par pré-conception : EXIG_EMTT_PUISSANCE EXIG_EMTT_INDICATEUR C1-22

- La carte émettrice possède un composant qui assure l'émission de trames protocolaires a la carte réceptrice sous émission infrarouge et chaque diode infrarouge doit être alimenté sous 200 mA crête pour respecter l'exigence:

Nous pouvons utiliser pour cela:

- 2 LEDs infrarouges, avec un angle et un écartement choisi, qui seront mis en série pour respecter et être en dessous de la valeur du régulateur linéaire.

- 1 Transistor pour alimenter et amplifier les LEDs sous 200mA



Le tout sera alimenté par un régulateur linéaire en 5V.

- La carte émettrice possède un indicateur lumineux informant l'utilisateur de la mise sous tension de la carte de couleur verte.
Solution trouvé :
1 LED électroluminescente Verte avec une résistance le tout alimenté par un régulateur linéaire 5 V.

2.1.4 Bloc énergie

Référence de préconception: PRC08

Exigences client vérifiées par préconception : EXIG_EMTT_ENERGIE C1-22

Pour alimenter l'émetteur de notre carte, on prend un accumulateur de type Lithium-Polymère 2S qui alimente en 7,4V.

Cet accumulateur permet accompagné d'un régulateur d'alimenter l'émetteur en 5V pendant au moins 60 minutes.

Afin de régulariser la tension à 5V pour alimenter l'ensemble de l'électronique de l'émetteur nous utiliserons un régulateur linéaire.

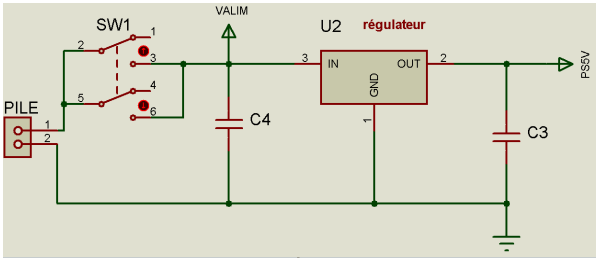
Nous avons choisi le LM 11 17 T-5 pour sa tension plus lisse +- 0.05V et son courant de sortie plus élevé.

Référence de préconception: PRC09

Exigences client vérifiées par pré-conception : EXIG_EMTT_INTERRUPTEUR C1-22

De manière à pouvoir mettre sous/hors tension le circuit électronique de l'émetteur, il faut installer un interrupteur.

Pour ce faire nous choisissons l'interrupteur JS 202011 SCQN pour sa simplicité d'utilisation, son coût et sa disponibilité.



2.2 Architecture Récepteur C1-03 C1-09 C1-10

Référence de préconception: PRC10

Exigences client vérifiées par préconception : EXIG_RCPT_CAPTEUR, EXIG_RCPT_TRAITEMENT, EXIG_RCPT_SECURITE, EXIG_RCPT_RETENTISSEMENT, EXIG_RCPT_INDICATEUR, EXIG_RCPT_CONNEXION, EXIG_RCPT_KLAXON, EXIG_RCPT_MOTEUR, EXIG_RCPT_ROUE, EXIG_RCPT_ENERGIE. C1-22

L'architecture du produit s'articule en 4 blocs

- Le bloc ACQUISITION D'INFORMATIONS comporte tous les composants d'interface machine.
- Le bloc TRAITEMENT D'INFORMATIONS comporte le coeur de traitement permettant de faire l'acquisition des données des interfaces homme machine (NEC), le traitement de ces données.
- Le bloc ACTIONS comporte tous les composants actifs.
- Le bloc ÉNERGIE comporte les organes fournissant l'alimentation et la régulation de la carte.

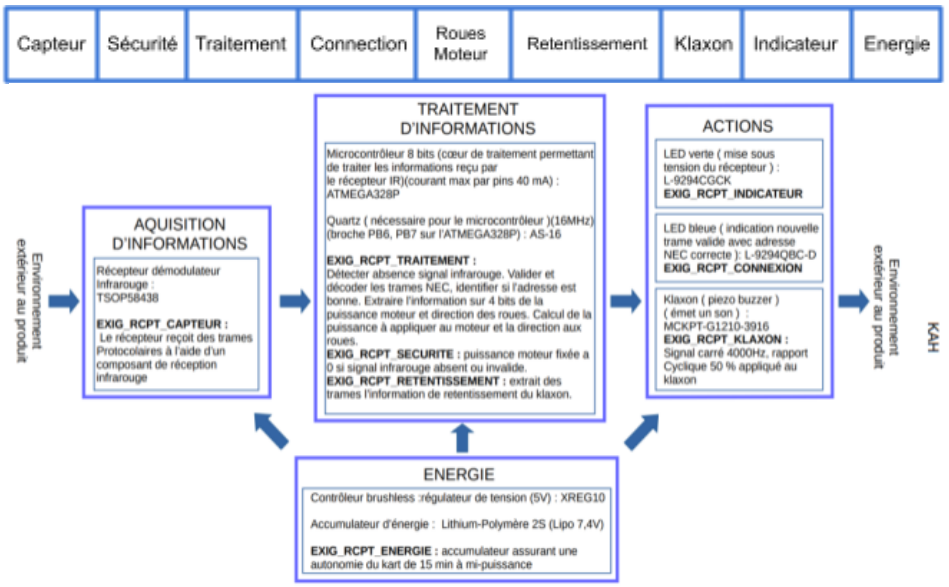


Figure 5: Schéma synoptique récepteur.

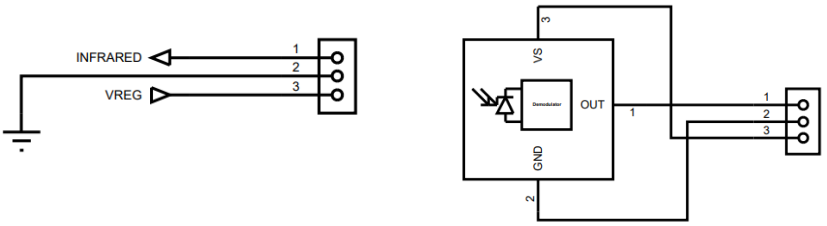
Le bloc acquisition reçoit des information NEC, émis par l'émetteur, à l'aide d'un capteur infrarouge. Celui-ci envoie cette information NEC dans le microcontrôleur de la partie traitement. Après traitement, différentes informations vont être envoyées afin de contrôler différents moteurs, une LED ou un buzzer. La partie énergie alimente tous ces blocs en fournissant une tension de 7,4 V convertit en 5 V. **C1-11**

2.2.1 Bloc acquisition

Référence de préconception: PRC11

Exigences client vérifiées par préconception : EXIG_RCPT_CAPTEUR C1-22

D'après l'exigence EXIG_RCPT_CAPTEUR : La carte possède un récepteur infrarouge pour récupérer les informations de l'émetteur. Nous déterminerons le capteur le plus adéquat afin d'assurer une bonne réception des informations émises par les LEDs infrarouges de l'émetteur.
Il est connecté à la carte à l'aide de connecteurs.



2.2.2 Bloc traitement

(MANTAUX Guéroc, BERNYER Quentin)

Référence de préconception: PRC12

Exigences client vérifiées par préconception : EXIG_RCPT_TRAITEMENT, EXIG_RCPT_SECURITE, EXIG_RCPT_RETENTISSEMENT C1-22

Afin de répondre aux exigences, nous avons choisi de prendre un microcontrôleur comme solution technique :

- L'ATMEGA328P serait une solution technique adaptée pour répondre aux exigences.
- D'après l'ensemble des exigences, et grâce au datasheet ce microprocesseur possède le bon nombre de broches I/O afin d'accueillir les différents périphériques.
- Étant le microcontrôleur le moins puissant en notre possession (fonctionne sur 8 bits), l'ATMEGA328P possède également d'autres atouts comme une consommation moindre et un prix réduit (voir nomenclature).

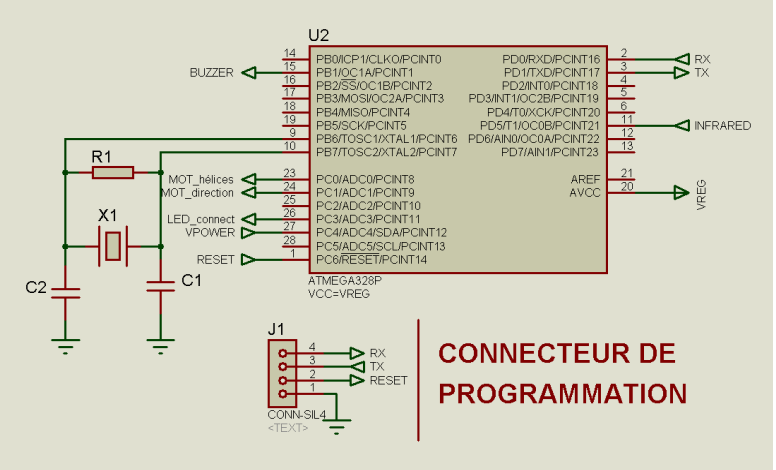


Figure 6 : Exemple branchement ATMEGA328P

2.2.3 Bloc action

Référence de préconception: PRC13

Exigences client vérifiées par préconception : EXIG_RCPT_INDICATEUR, EXIG_RCPT_CONNEXION, EXIG_RCPT_KLAXON, EXIG_RCPT_MOTEUR, EXIG_RCPT_ROUE C1-22

D'après l'exigence EXIG_RCPT_INDICATEUR **:** la carte possède une LED verte afin d'indiquer que le récepteur est sous tension. Pour cela nous avons sélectionné une LED de couleur verte utilisé en modélisme.

D'après l'exigence EXIG_RCPT_CONNEXION : la carte possède une LED bleue afin d'indiquer à l'utilisateur que le récepteur à reçu une nouvelle trame infrarouge valide. Nous avons allors choisis une LED de couleur bleue qui est généralement utilisé pour cette fonction en modélisme.

D'après l'exigence EXIG_RCPT_KLAXON : la carte possède un composant sonore qui permettra au kart de klaxonner. Pour cela nous avons choisi un buzzer.

D'après l'exigence EXIG_RCPT_MOTEUR : un signal PWM est créé grâce au microcontrôleur. Celui-ci servira à piloter un moteur qui servira à faire tourner les hélices. Pour ce faire, nous avons choisi un contrôleur de moteur brushless qui est branché directement sur la batterie et au microcontrôleur afin de moduler le signal de sortie pour faire fonctionner un moteur brushless.

D'après l'exigence EXIG_RCPT_ROUE : un signal PWM est créé par le microcontrôleur afin de contrôler la direction des roues. Nous avons donc choisi un servomoteur qui intègre directement un contrôleur et un moteur.

2.2.4 Bloc énergie

(MANTAUX Guéroc, BERNYER Quentin)

Référence de préconception: PRC14

Exigences client vérifiées par préconception : EXIG_RCPT_ENERGIE C1-22

- Pour alimenter l'émetteur de notre carte, nous avons choisi un accumulateur de type Lithium-Polymère 2 qui alimente la carte en 7,4V.
- Cet accumulateur doit être accompagné d'un régulateur qui permet d'alimenter le récepteur en 5V pendant au moins 15 minutes.
- Afin de régulariser la tension à 5V pour alimenter l'ensemble de l'électronique du bloc récepteur nous utiliserons un régulateur linéaire :

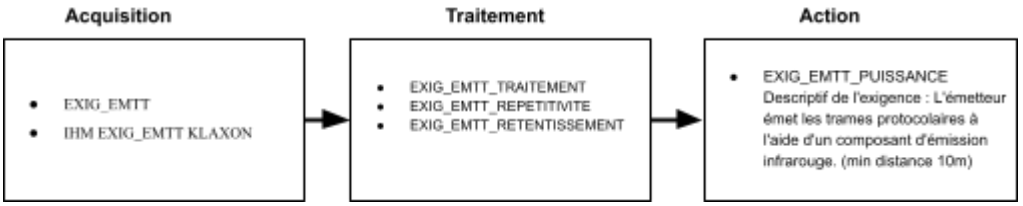
  Nous avons choisi le LM 11 17 T-5 pour sa tension plus lisse +- 0.05V et son courant de sortie plus élevé.

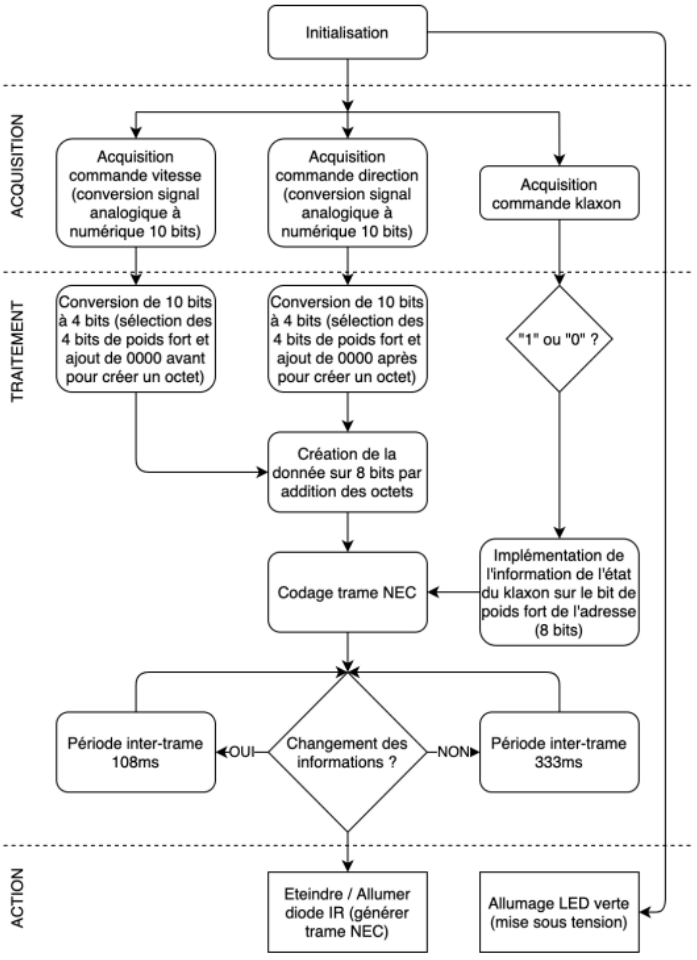
3 Architecture Informatique

3.1 Architecture Émetteur

Figure 7 et 8.

Architecture informatique émetteur.
(Exigences)





Logigramme programme émetteur

3.1.1 Bloc acquisition

Référence de préconception: PRC15

Exigences client vérifiées par préconception : EXIG_EMTT_IHM EXIG_EMTT_KLAXON C1-22

- 1 potentiomètre pour les roues :

branché sur broche I/O du MCU : sa valeur sera déchiffrée grâce à la fonction analogRead.

- 1 potentiomètre pour l'hélice :

branché sur broche I/O du MCU : sa valeur sera déchiffrée grâce à la fonction analogRead.

- 1 bouton poussoir pour le klaxon :

branché sur broche I/O du MCU : sa valeur sera déchiffrée grâce à la fonction digitalRead.

- La fonction analogRead permet au microcontrôleur de lire en entrée une donnée analogique afin de la traduire en donnée numérique traitable par le processeur.
  Elle lit la valeur de la tension présente sur la broche spécifiée.Les broches sont connectées à un convertisseur analogique-numérique 10 bits. Cela signifie qu'il est possible de transformer la tension d'entrée entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023.

  Il en résulte une résolution (écart entre 2 mesures) de : 5 volts / 1024 intervalles, autrement dit une précision de 0.0049 volts (4.9 mV) par intervalle !

  Une conversion analogique-numérique dure environ 100 µs pour convertir l'entrée analogique, et donc la fréquence maximale de conversion est environ de 10 000 fois par seconde.

  Syntaxe : analogRead(broche_analogique)

- La fonction DigitalRead permet au microcontrôleur de lire l'état (= le niveau logique) d'une broche précise en entrée numérique, et renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais).
  Syntaxe : digitalRead(broche)

Rappel :
Pour pouvoir lire une broche en tant qu'entrée numérique :

- configurer cette broche en entrée avec l'instruction pinMode(broche, INPUT);

- activer le "rappel au plus" interne dans le cas d'un bouton poussoir avec l'instruction digitalWrite(broche, HIGH)

ces fonctions logicielles sont intégrées à l'ATMEGA 328P, qui est codé en C

3.1.2 Bloc traitement

Référence de préconception: PRC16

Exigences client vérifiées par préconception : EXIG_EMTT_TRAITEMENT, EXIG_EMTT_REPETITIVE, EXIG_EMTT_RETENTISSEMENT C1-22

- Il est nécessaire d'inclure les bibliothèques : #include <arduino.h>;
- On lit la valeur des 2 potentiomètres en les récupérant dans le bloc acquisition.
- On utilise ensuite la fonction :

map(valeur a transformé , min , max , nouveaux min , nouveaux max) ;

Cette fonction modifiera la plage de valeur afin d'avoir une valeur sur 4 bits, soit nouveaux min = 0 et nouveaux max = 15.

On assemble alors les valeurs de 4 bits obtenues afin de former 1 octet pour créer une trame NEC.

On utilise l'opération bitshift ('variable << nombre de décalage') qui décale d'un nombre de variable définie vers la gauche. On utilisera cette fonction avec un décalage de 4, afin de pouvoir décaler nos 4 bits les plus faibles sur les 4 bits les plus forts. Une fois nos 4 premier bits écrit et décalé on utilise une fonction et ('&') afin de conserver les valeurs écrites et rajouter les valeurs de notre deuxième mots.

Pour créer une fonction d'émission d'une trame NEC on utilise la fonction nécessaire afin de contrôler les LEDs infrarouges :

void GenererTrameNEC(int Broche, unsigned char Adresse, unsigned char Donnee);
Broche correspond à la broche Arduino sur laquelle est connectée la LED d'émission infrarouge,
Adresse est la valeur de l'adresse NEC à transmettre dans la trame NEC,
Donnée est la valeur de la donnée NEC à transmettre dans la trame NEC.

D'après l'exigence EXIG_EMTT_REPETITIVE : il est demandé d'émettre une nouvelle trame à chaque fois qu'il y a une nouvelle donnée. En revanche, si la donnée est égale à la précédente, la période d'émission des trames est fixée à 333 ms (+/- 10%).

On peut alors écrire la structure du programme :
Si nouvelle trame =! trame précédente :
    nouvelle trame NEC
Si elles sont identiques :
    période d'émission = 333 ms +/- 10%

Pour contrôler le temps on utilise la fonction : delay(); ou delayMicroseconds();

D'après l'exigence EXIG_EMTT_RETENTISSEMENT : il faut que les trames intègrent l'information que le bouton poussoir est appuyé.

On récupère l'information de la variable du bouton poussoir fourni dans le bloc acquisition pour savoir son état. On convertit alors cette valeur en trame NEC. Pour ce faire on utilise la fonction void GenererBit0NEC(int Broche); ou void GenererBit1NEC(int Broche);

3.1.3 Bloc action

Référence de préconception: PRC17

Exigences client vérifiées par préconception : EXIG_EMTT_PUISSANCE EXIG_EMTT_INDICATEUR C1-22

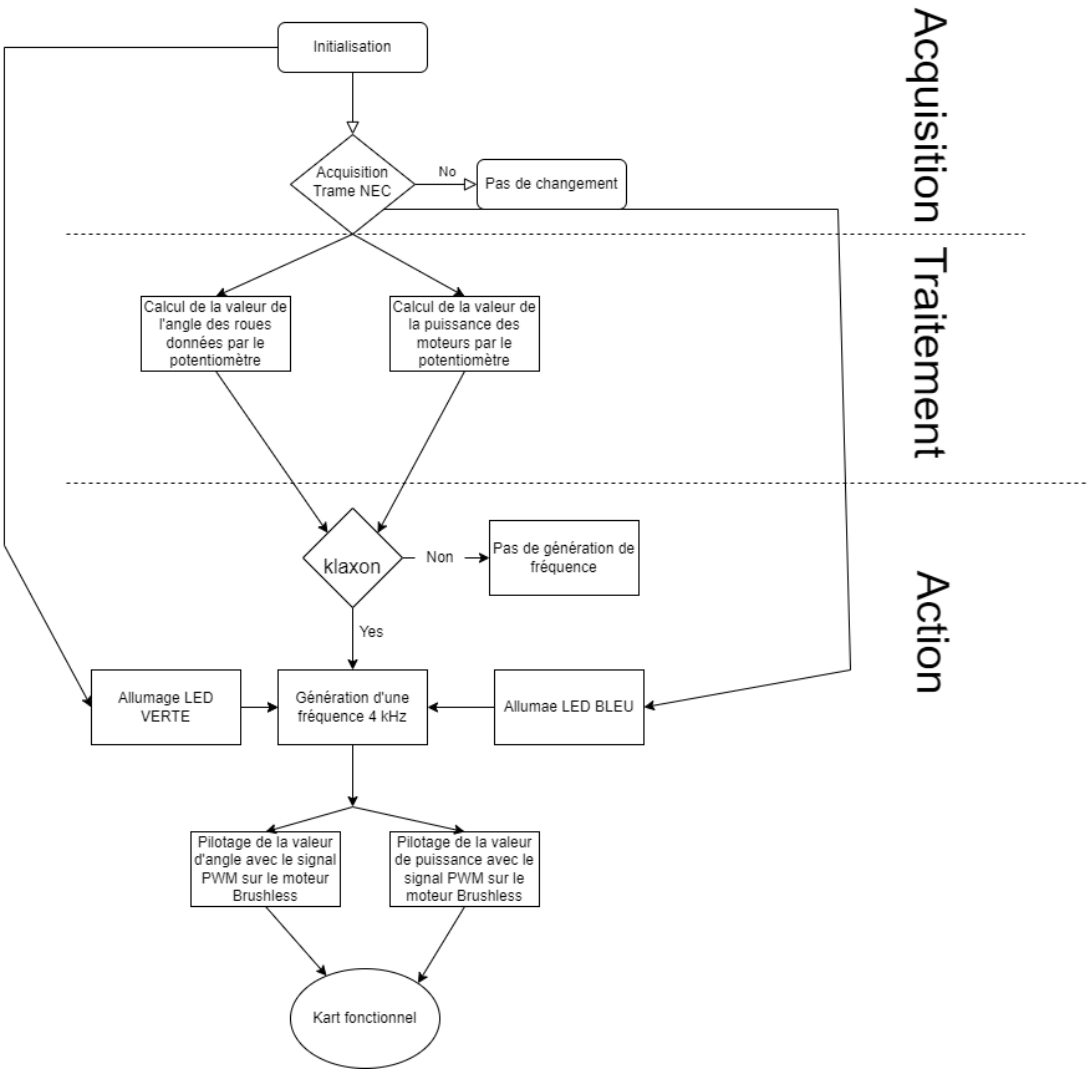
- les LEDs infrarouges seront commandées par le microcontrôleur grâce à la fonction suivante :

  void GenererTrameNEC(int Broche, unsigned char Adresse, unsigned char Donnee);

Cette fonction sert à émettre une trame NEC, qui sera nécessaire afin de contrôler les LEDs infrarouges pour envoyer les bonnes informations. La fonction GenererTrameNEC est la fonction principale. Elle peut être complétée avec différentes sous fonctions pour faire les tâches demandées

Ces fonctions font partie de la bibliothèque <arduino.h> . pour la paramétrer correctement, il faut rentrer GenererTrameNEC(broche LED, adresse récepteur, donnée à envoyer)

3.2 Architecture Récepteur



3.2.1 Bloc acquisition

Référence de préconception: PRC18

Exigences client vérifiées par préconception : EXIG_RCPT_CAPTEUR C1-22

Nous devrons utiliser les bibliothèques suivantes pour recevoir les trames.

```
#include <arduino.h>
#include <NEC.h>
```

La bibliothèque NEC.h ajoute la fonction "AcquerirTrameNEC" qui retourne une valeur selon l'état de la trame reçue.

- 0 si la trame est valide.
- 1 en cas d'erreur de réception de la trame NEC.
- 2 si aucune trame est reçue.

La fonction s'utilise comme ci- dessous.

```
char AcquerirTrameNEC(int Broche, unsigned char* Adresse, unsigned char* Donnée);
```

- "Adresse" retourne la valeur de l'adresse NEC incluse dans la trame NEC reçue
- "Donnée" retourne la valeur de la donnée NEC incluse dans la trame NEC reçue

3.2.2 Bloc traitement

Référence de préconception: PRC19

Exigences client vérifiées par préconception : EXIG_RCPT_TRAITEMENT EXIG_RCPT_SECURITE EXIG_RCPT_RETENTISSEMENT C1-22

Si la trame NEC acquise n'est pas différente de 0 et que l'adresse et le bit 7 sont égale à l'adresse NEC. Alors la trame est considérée valide.

On pourra utiliser la commande suivante:
map(valeur a transformé Déplacement , min , max , nouveaux min , nouveaux max) ;
Déplacement avec >> x on déplace les bits vers les bits de poid faible en faisant apparaître des 0 de x fois.
On fait de même avec << qui décale vers le bits de poid fort
map nous permet de définir une plage de valeur la rendant de 0 à 1023 vers 0 à 180 par exemple.

-Nous pourrons utiliser la commande d'arrêt du moteur brushless montré en partie action ci-dessous pour respecter l'exigence Sécurité.

3.2.3 Bloc action

Référence de préconception: PRC20

Exigences client vérifiées par préconception :EXIG_RCPT_MOTEUR : EXIG_RCPT_ROUE EXIG_RCPT_CONNEXION EXIG_RCPT_KLAXON C1-22

Les fonctions suivantes fonctionne pour le servo moteur et le moteur brushless

A l'aide de la bibliothèque servo.h

Servo myservo ; permet de déclarer une variable de type Servo, avant setup

myservo.attach(9) ; attache la variable myservo à la broche 9, dans le setup

Servo.write(xx) ; permet de définir la vitesse ou l'angle suivant le moteur.

- quand xx est à 0, 100% de vitesse dans un sens ou à 0°
- quand xx est à 180, 100% de vitesse dans l'autre sens ou a 180 °

-Pour contrôler un buzzer nous on peut utilisé:

pinMode(broche , OUTPUT);définir l'utilisation de la broche

tone (broche, Fréquence d'émission); Permet de faire fonctionner le buzzer.

notone(broche); Permet d'éteindre le buzzer.

-Nous pouvons commander une led grâce à:

pinMode(ledPin, OUTPUT) ; définition de l'utilisation et de la broche

digitalWrite(ledPin, HIGH) ; led allumé

digitalWrite(ledPin, LOW) ; led éteinte

4 Nomenclature

Composants	Prix (unité)	Quantité
Atmega328p MCU	1,85	2
Quartz	0,1	1
régulateur linéaire	1,58	1
accumulateur	18,86	1
interrupteur	0,34	1
TSAL6200 diodeinfra	0,31	2
PTA 4543 Potentiomètre	1,29	2
Led verte	0,15	2
Capteur infrarouge	0,63	1
LED bleue	0,25	1
Buzzer	0,65	1
Contrôleur de moteur brushless	7,55	1
Moteur brushless	12	1
ServoMoteur	15	1
Total	64.16	

III -Conception détaillée du produit

1 . Architecture électronique

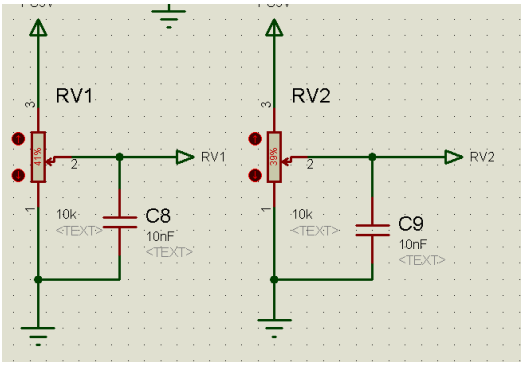
1.1. Architecture électronique carte émetteur

1.1.1 Etage 1 Architecture Acquisition (Petitjean Nathan, Pelamatti Paul) C1-04 C1-09 C1-21 C1-24 C1-25 C1-26

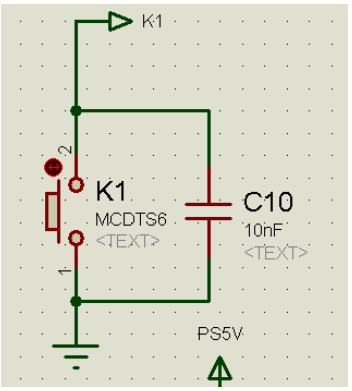
Référence de conception : CCPT01

Exigences client vérifiées : EXIG_EMTT_IHM, EXIG_EMTT_ KLAXON C1-22

- D'après l'exigence EMTT_IHM : la carte doit disposer de deux potentiomètres permettant de faire varier l'angle de rotation des roues du kart ainsi que la puissance moteur. Nous avons choisi d'utiliser 2 potentiomètres rectilignes (PTA4543), qui sont des résistances variables de 10KΩ qui nous permettront de faire varier la tension pour qu'elle soit ensuite récupérée par l'ATMEGA328P. Et chacun est associé à un condensateur de 10nF permettant d'enlever les éventuelles perturbations et les parasites lors du déplacement du curseur.



- Pour répondre à l'exigence : EMTT_ KLAXON. Le bloc acquisition possède également un bouton poussoir MCDTS2 permettant d'activer le klaxon, il est lui aussi muni d'un condensateur. Nous allons utiliser la résistance interne du microcontrôleur en partie traitement qui est relié à la broche de l'ATMEGA 328P pour éviter une consommation de courant trop élevé tout en étant inférieur à la valeur résistive de la peau humaine évitant ainsi d'éventuels court-circuits lors des manipulations de la carte . Nous obtenons un '0' logique quand le bouton n'est pas appuyé et un '1' quand celui-ci est actionné.



1.1.2 Etage 2 Architecture Traitement (Dumas Adrien) C1-04 C1-09 C1-21 C1-24 C1-25 C1-26

Référence de conception : CCPT02

Exigences client vérifiées : EXIG_EMTT_TRAITEMENT, EXIG_EMTT_RETENTISSEMENT, EXIG_EMTT_REPETITIVITE C1-22

Le microcontrôleur est un composant électronique qui permet de prendre plusieurs signaux d'informations en entrée afin de pouvoir les traiter. Ces données une fois traitées peuvent ensuite servir à donner des instructions en sortie.

Nous avons choisi cette l'ATMEGA 328P 28 PDIP comme solution technique pour répondre :

- Aux exigences EXIG_EMTT_TRAITEMENT et EXIG_EMTT_RETENTISSEMENT. Le microcontrôleur permet d'acquérir des informations analogiques et numérique en provenance des potentiométriques et du bouton klaxon grâce à des broches analogique (broche 23 à 28) et digital (broche 1 à 6, 9 à 19 et 23 à 28), de numériser les informations analogiques en informations numériques à l'aide de son ADC (10 bit), d'encoder les données numériques de puissances et directions sur 4 bits chacune pour créer un octet de donnée afin de construire les trames de transmission conformément au protocole NEC grâce à son coeur de traitement 8 bits.

- A l'exigence EXIG_EMTT_REPETITIVITE. Le microcontrôleur émet grâce à son cœur de traitement par une broche digital une trame NEC à chaque fois que les données sont nouvelles et ceci sans délai autre que le délai minimum inter-trame imposé par le protocole NEC, si les données de trames NEC consécutives sont identiques, la période entre deux émissions est fixée à 333 millisecondes (+/- 33 millisecondes).

- A l'exigence EXIG_EMTT_ENERGIE. Le microcontrôleur ATMEGA328P a une consommation basse, avec un maximum de 12 mA pour une tension d'alimentation de 5V.
  il peut fournir 40 mA max par pins et 200 mA max au total des pins
  l'ATMEGA fournit en courant
  la led verte avec 10 mA
  le total des deux potentiomètres avec 2mA
  pour le RESET avec 0,5mA
  le résonateur avec 5mA
  Il fournit donc 17,5 mA ce qui est bien inférieur à ce qu'il peut fournir.

- A l'exigence EXIG_COÛT. Le microcontrôleur ATMEGA328P est le moins cher parmi ceux proposés (voir nomenclature).

Résonateur :

Le microcontrôleur ATMEGA328P à besoin conformément à EXIG_EMTT_REPETITIVITE d'un rythme régulier et très précis.

Nous avons donc choisit d'utiliser pour cela un Résonateur ZZT 16.00 MX cadencé par un quartz interne(AS-16.000-18) à 16MHz

Le résonateur à pas besoin d'un montage particulier, il est directement branché sur les broches XTAL1 et XTAL2 du microcontrôleur (broches de clock).

Reset :

Le microcontrôleur ATMEGA328P doit pouvoir lire son code aux valeurs initiales.

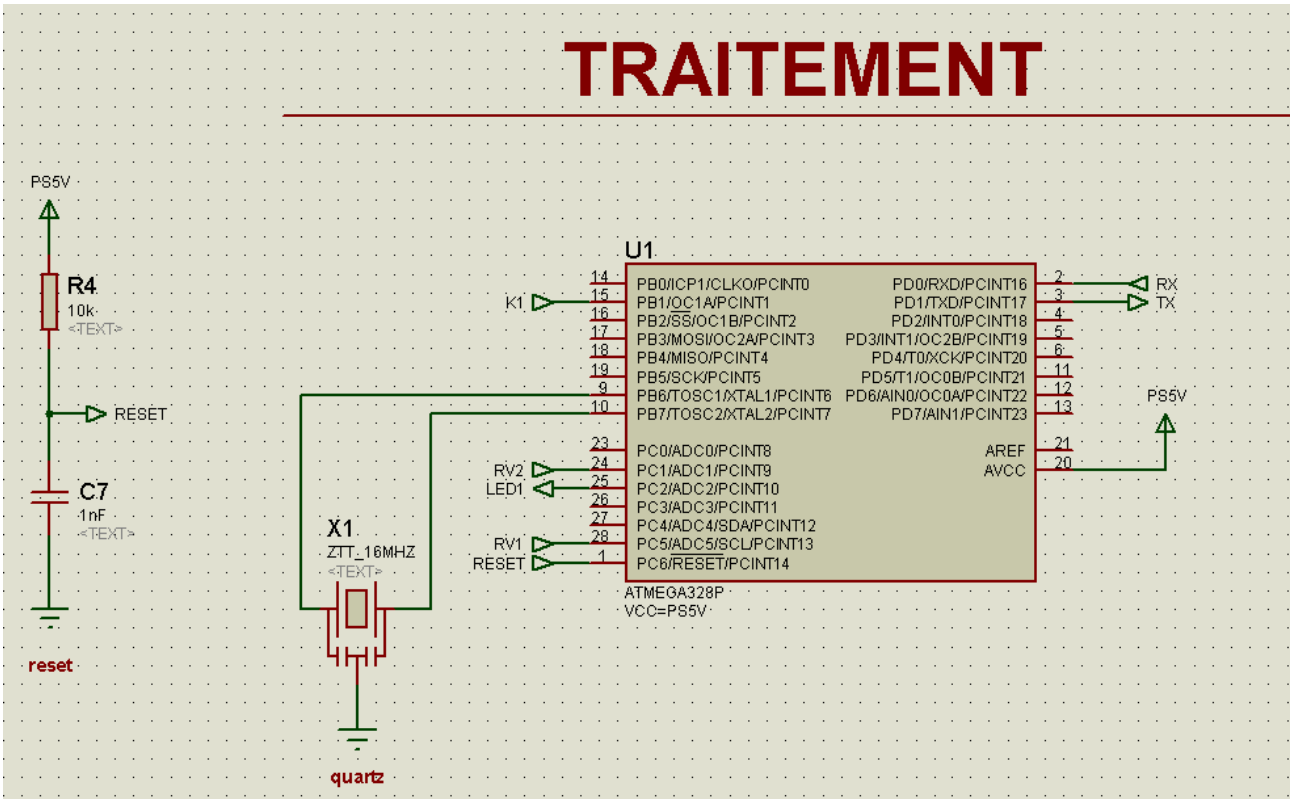
Pour ça on ajoute un montage RESET branché sur la broche RESET (broche 1).

Ce montage est composé d'une résistance R4 =10 kΩ (pour atteindre le seuil de reset) et d'un condensateur C7 = 1nF (lisse le signal) en série branché au PS5V afin d'effectuer un Power-On Reset et réinitialiser les valeurs à la mise sous tension de la carte.

Connecteur de programmation :

Pour pouvoir communiquer avec le microcontrôleur l'ATEMEGA 328P il faut lui connecter un connecteur de programmation J1 (4 pins), branché sur les broches ; RX, TX, RESET et le GND .

Cette communication sert à programmer le microcontrôleur ou à vérifier son bon fonctionnement.



1.1.3 Etage 3 Architecture Action (Petitjean Nathan, Pelamatti paul) C1-04 C1-09 C1-21 C1-24 C1-25 C1-26

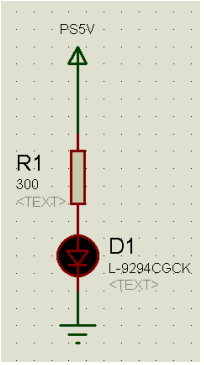
Référence de conception : CCPT03

Exigences client vérifiées : EXIG_EMTT_PUISSANCE EXIG_EMTT_INDICATEUR C1-22

- Pour répondre à l'exigence : EMTT_ INDICATEUR. Le bloc Action possède une led (L-9294 CGCK) alimenté grâce au régulateur linéaire 5V, avec sa résistance associée qui est de 300Ω E24 déterminée grâce aux calcul suivant:

  on prend 10 mA pour la led

  $R1 = \frac{5-2}{0{,}010} = 300\ \Omega$ donc 300 E24 +/- 5%



- Pour répondre à l'exigence : EMTT PUISSANCE.

  Le bloc Action possède deux diodes infrarouge mises en série avec une résistance R2 20Ω et un transistor pour pouvoir commander les diodes. Le tout est alimenté par la batterie en 7,4 V. Le transistor a une résistance R3 430Ω entre sa base et le microcontrôleur permettant de limiter la tension à ses bornes

  Vce = 0,7 V tension seuil led 1,45V

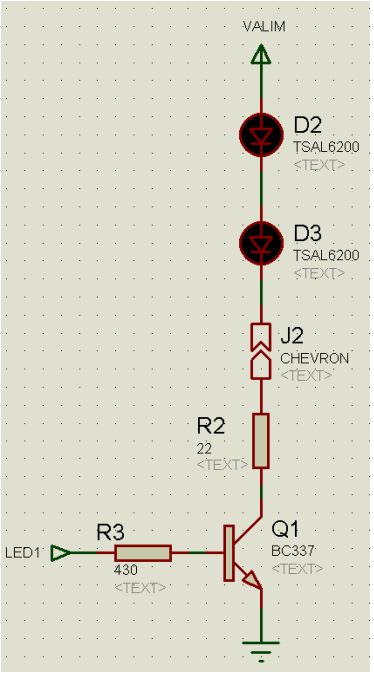
  $R2 = \frac{7,4-0,7-2*1,45}{0,2} = \frac{3,8}{0,2}$=19 Ω donc 20 ohms E24+/-5%

  La résistance permet d'avoir 200 mA en pic.

  Vbe = 0.6 V

  R3= (5 - 0.6) / 0,010 = 440 ohms donc 430 e 24 +/- 5 %

  Nous avons aussi ajouté un chevron permettant de mesurer le courant passant dans les led infrarouge.



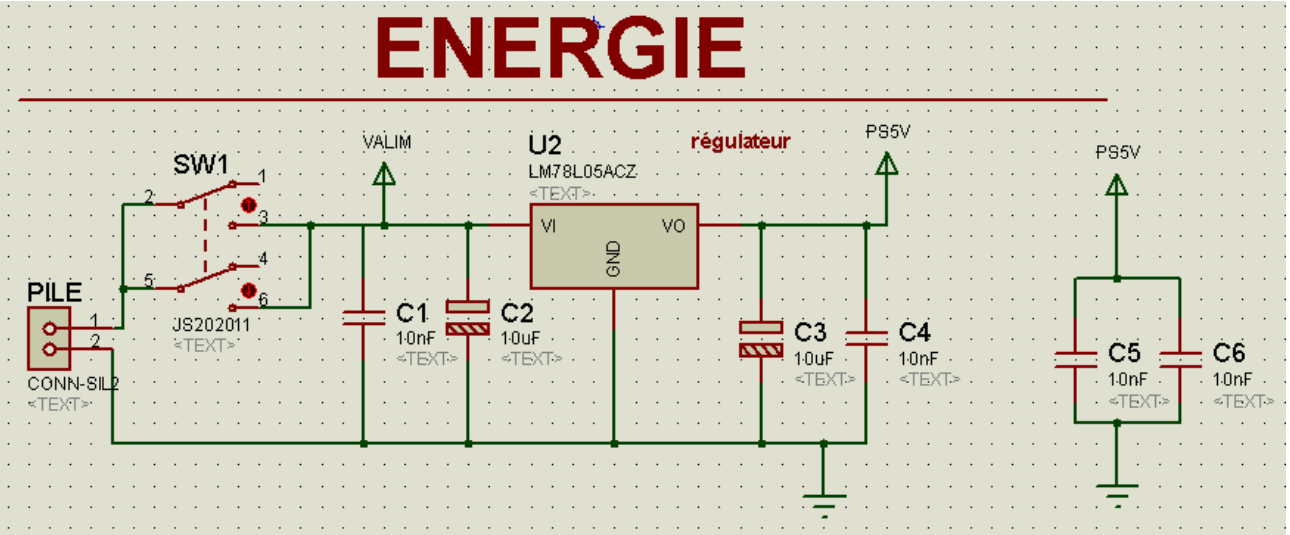
1.1.4 Etage 4 Architecture Energie (Dumas Adrien) C1-04 C1-09 C1-21 C1-24 C1-25 C1-26

Référence de conception : CCPT04

Exigences client vérifiées : EXIG_EMTT_ENERGIE, EXIG_EMTT_INTERRUPTEUR

Afin d'alimenter le montage de la carte émetteur nous utiliserons une batterie et .

Nous avons choisi une batterie LIPO 2S 350mAh qui alimente en 7.4V comme solution technique pour répondre :



- A l'exigence EXIG_EMTT_ENERGIE.
  La batterie doit offrir à la carte une autonomie de 60 minutes.

  10 mA pour la led
  200 mA pour les led infra rouge
  2mA pour le total des deux potentiomètres
  12 mA pour le microcontrôleur
  0,5 mA pour le RESET
  5 mA pour le résonateur
  6 mA pour le régulateur
  235,5 mA de consommation total

  par sécurité on considère les batterie à 80% de charge 0.8

  AUTO = (capacité_batterie* 0.8) / consommations_total

  pour la batterie 350 mA/h AUTO = 1.2 heures = 71 minutes

  pour la batterie 500 mA/h AUTO =1.7 heures = 101 minutes

  pour la batterie 1000 mA/h AUTO =3.4 heures = 203 minutes

  Les trois capacités de batteries sont conformes mais nous avons choisi la LIPO 2S 350mAh pour son coût moins élevé et son plus petit encombrement.

  - Le circuit doit être alimenté en 5V on utilise pour cela un régulateur de tension 5V le LM78 se régulateur doit alimenter le microcontrôleur et les circuit qui lui sont rattaché avec un courant max de 40 mA

le atmega consomme maximum 12 mA et il alimente les circuits auquel il est rattaché à hauteur de 17,5 mA le courant fournit par le régulateur et donc suffisant.

Afin de mettre sous et hors tension le circuit nous prenons un interrupteur JS2020

1.2 Architecture électronique carte récepteur

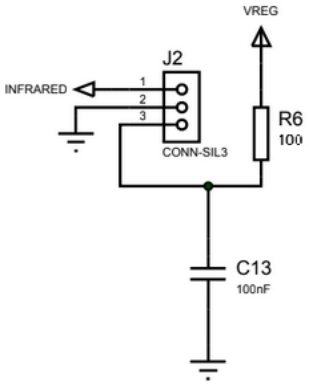
1.2.1 Bloc acquisition C1-04 C1-09 C1-21 C1-24 C1-25 C1-26

(HORTOS Sullivan, CAZADE Amory)

Référence de conception : CCPT05

Exigences client vérifiées : EXIG_RCPT_CAPTEUR C1-22

D'après l'exigence EXIG_RCPT_CAPTEUR : Le bloc acquisition de la carte possède un récepteur infrarouge afin de récupérer les informations fournies par l'émetteur. Nous avons alors choisi le capteur infrarouge TSOP4438. Il est connecté à la carte à l'aide d'un connecteur.



Nous avons choisi ce capteur, car il a une plus grande zone de détection. Il permet de détecter le signal dans une demi sphère et cela jusqu'à 10m. Celui-ci est utilisé avec une résistance de 10 Ω ainsi qu'un condensateur de 100nF.

La broche 1 est reliée au microcontrôleur afin d'envoyer l'information acquise par le capteur infrarouge au microcontrôleur.
La broche 2 est reliée à la masse.
La broche 3 est reliée au régulateur linéair (qui se situe dans le contrôleur de moteur brushless) afin de ne pas recevoir une tension supérieure à 5,5 V.

Nous avons utilisé la datasheet afin de déterminer la résistance et le condensateur associé. C1-23

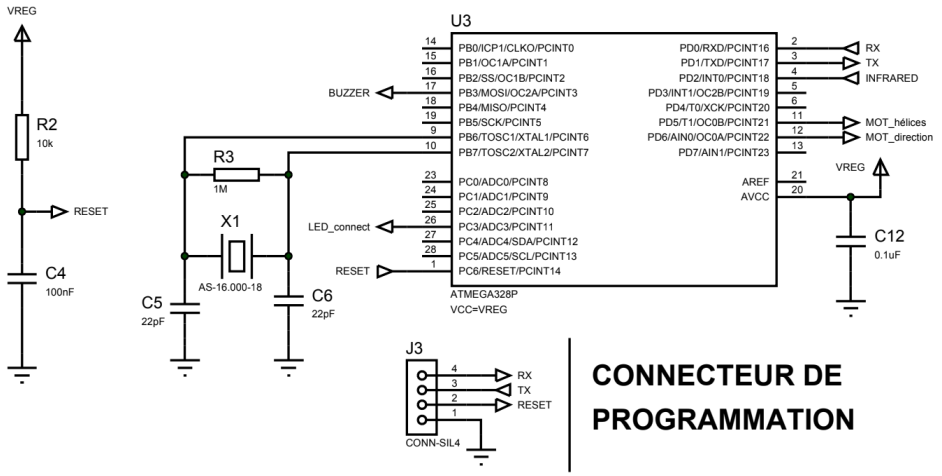
1.2.2. Bloc traitement C1-21 C1-24 C1-25 C1-26

(MANTAUX Guéroc, BERNYER Quentin)

Référence de conception : CCPT06

Exigences client vérifiées : EXIG_RCPT_CAPTEUR, EXIG_RCPT_TRAITEMENT, EXIG_RCPT_SECURITE, EXIG_RCPT_RETENTISSEMENT, EXIG_RCPT_INDICATEUR, EXIG_RCPT_CONNEXION, EXIG_RCPT_KLAXON, EXIG_RCPT_MOTEUR, EXIG_RCPT_ROUE, EXIG_RCPT_ENERGIE. C1-22

Le microcontrôleur est un composant électronique qui permet de prendre plusieurs signaux d'informations en entrée afin de pouvoir les traiter. Ces données une fois traitées peuvent ensuite servir à donner des instructions en sortie.



Nous avons choisi cette l'ATMEGA 328P comme solution technique pour répondre :

- A l'exigence EXIG_RCPT_TRAITEMENT, car ce microcontrôleur permet de détecter la présence de signaux infrarouges grâce à des périphériques externes (broche 4 sur schéma, qui permet de recevoir des signaux numériques). il peut décoder des trames NEC envoyées par un émetteur, grâce à ses broches I/O. l'ATMEGA peut également contrôler la validité de ses trames, étant un processeur fonctionnant sur 8 bits, il est conforme avec les commandes envoyées sur 8 bits par l'émetteur, comme la direction des roues (broche 12 sur schéma) et la puissance du moteur (broche 11 sur schéma). Les broches sélectionnées pour les moteurs (11 et 12) broches analogiques, qui permettent d'envoyer des signaux analogiques aux moteurs.

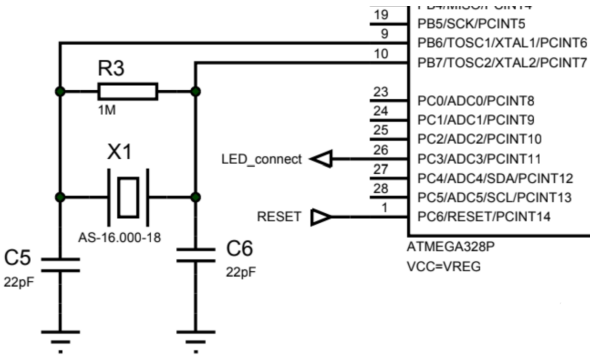
- A l'exigence EXIG_RCPT_SECURITE, car ce microcontrôleur permet de décoder et de détecter la présence de signaux infrarouges grâce à des périphériques externes (broche 4 sur schéma). Il peut donc détecter l'absence de signal, ou comparer l'adresse donnée avec celle attendue, et donc produire un signal de sortie nul (broche 11 sur schéma) si le résultat n'est pas celui attendu (pas de signal ou signal non valide).

- A l'exigence EXIG_RCPT_RETENTISSEMENT car ce microcontrôleur possède des broches I/O qui permettent de détecter des signaux numériques (broche 4 sur schéma). il peut donc décoder ce signal et transmettre en sortie une tension permettant d'alimenter ou non un buzzer en fonction de la trame donnée (broche 17 sur schéma, qui prends en compte la fonction Tone, permettant de faire fonctionner le buzzer a certaines fréquences)

- A l'exigence EXIG_RCPT_ENERGIE de par sa consommation moindre (12mA max). Cette exigence nous impose une lipo 2S. Et une autonomie définie à 15min à mie puissance. La carte à une tension d'entrée max de 5,5 V elle sera donc alimenté par le régulateur 5V du contrôleur moteur XREG.
  Calcul courant de sortie consommée par le système sur la carte : 1 + 5 + 2 = 8 mA
  Cela est bien inférieur au courant au courant maximum total de la carte :
  max per pin = 40 mA.
  max total pins = 200 mA.
- A l'exigence EXIG_COÛT grâce à son prix réduit (voir nomenclature).

Etage Quartz :

Étant le processeur le moins puissant en notre possession (fonctionne sur 8 bits à 20mHz max.).



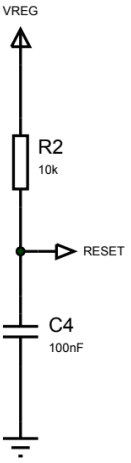
Il sera ici cadencé à 16mHz par un quartz (voir schéma).

Nous avons choisi d'utiliser le quartz AS-16.000-18 qui nécessite deux condensateurs de 22pF en série et une résistance R3 de 1MΩ (recommandation constructeur)..

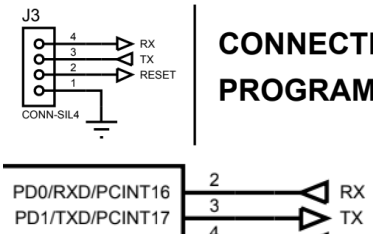
Le quartz a été branché en suivant les instructions constructeur, sur les broches XTAL1 et XTAL2 du microcontrôleur (broches de clock).

Etage Reset :

L'ATEMEGA 328P nécessite un reset pour pouvoir réinitialiser le la lecture du code aux valeurs initiales dans le micro contrôleur. Cet étage est composé d'une résistance et d'un condensateur, reliés au VREG afin d'effectuer un Power-On Reset et réinitialiser les valeurs à la mise sous tension de la carte. Une resistance de 10kΩ permet d'atteindre le seuil de 0.2*VCC (seuil de reset) et le condensateur C4 permet d'avoir un signal non bruité pour un reset fiable.



Etage Broche de communication :



Pour communiquer avec le microcontrôleur l'ATEMEGA 328P possède des broche série de RX et TX. La communication avec le microcontrôleur sert par exemple à sa programmation. La carte électronique possède donc un connecteur branché sur les pins de programmation permettant de reprogrammer cette dernière.

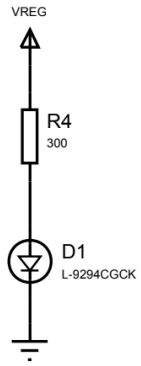
1.2.3. Bloc action C1-04 C1-09 C1-21 C1-24 C1-25 C1-26

(HORTOS Sullivan, CAZADE Amory)

Référence de conception : CCPT07

Exigences client vérifiées : EXIG_RCPT_INDICATEUR, EXIG_RCPT_CONNEXION, EXIG_RCPT_KLAXON, EXIG_RCPT_MOTEUR, EXIG_RCPT_ROUE C1-22

D'après l'exigence EXIG_RCPT_INDICATEUR **:** la carte possède une LED verte afin d'indiquer à l'utilisateur que le récepteur est sous tension. Pour cela nous avons sélectionné la LED de couleur verte L-9294CGCK qui est généralement utilisé, en modélisme, pour indiquer que le récepteur est sous tension.



Celle-ci est reliée directement à la batterie afin de s'allumer lorsque le Kart est sous tension.

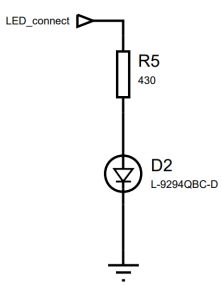
Une résistance est associée à cette LED :

On a un courant de 10 mA et une tension de 3 V.

Soit $R = \frac{U}{I} = \frac{3}{10.10^{-3}} = 300\ \Omega$

Nous avons alors choisi une résistance normalisée 300 Ω série E24

D'après l'exigence EXIG_RCPT_CONNEXION : la carte possède une LED bleue afin d'indiquer à l'utilisateur que le récepteur à reçu une nouvelle trame infrarouge valide. Nous avons alors choisi LED de couleur bleue L-9294QBC-D.



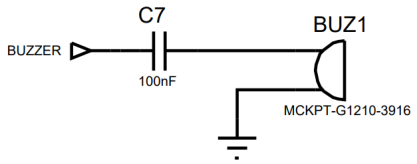
On a choisi de limiter le courant en mettant un courant de 5 mA . On a alors une tension de 2,1 V.

$R = \frac{U}{I} = \frac{2{,}1}{5.10^{-3}} = 420\ \Omega$

On prend une résistance normalisée série E24 430 Ω.

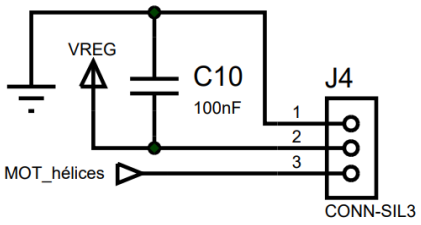
Pour dimensionner les résistances associées au LED, nous avons utilisé la loi d'ohm vu en électronique. **C1-23**

D'après l'exigence EXIG_RCPT_KLAXON : la carte possède un composant sonore qui permettra au kart de klaxonner. Pour cela nous avons choisi le buzzer MCKPT-G1210-3916.
Ce buzzer est associé à un condensateur pour lisser la tension carré qui arrive et ainsi ne pas avoir d'interférence.



On a choisi ce buzzer car celui-ci possède une fréquence de résonance de 4,5 kHz (+/- 0,5 Hz) ce qui est assez proche de notre signal délivré (4 kHz (+/- 100Hz)).

D'après l'exigence EXIG_RCPT_MOTEUR : un signal PWM est créé grâce au microcontrôleur. Celui-ci servira à piloter un moteur qui servira à faire tourner les hélices. Pour ce faire, nous avons choisi le contrôleur de moteur brushless XREG6 qui est branché directement sur la batterie et au microcontrôleur afin de moduler le signal de sortie pour faire fonctionner un moteur brushless.



Ce contrôleur possède un régulateur linéaire et offre ainsi une tension de 5 V à tous les composants ayant besoin d'une tension inférieure ou égale à 5 V. Il sort également un courant de 0,8 A.

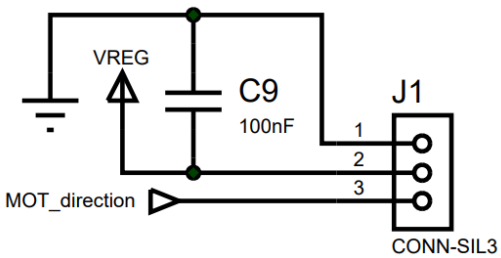
2 câbles sont branchés sur la batterie et 1 câble est branché au microcontrôleur afin de recevoir une information.

Un condensateur de découplage de 100 nF est placé entre l'alimentation et la masse.

Le moteur brushless associé et le Turgnigy 28-22-CQ 1400Kv Brushless Outrunner. Il accepte une tension max 9 V et un courant max de 5 A.

D'après l'exigence EXIG_RCPT_ROUE : un signal PWM est créé par le microcontrôleur afin de contrôler la direction des roues. Nous avons donc choisi le servomoteur Hitec HS322HD qui intègre directement un contrôleur et un moteur.

Il accepte une tension de 4,8 V à 6 V en sortie.



Un condensateur de découplage de 100 nF est placé entre l'alimentation et la masse.

1.2.4. Bloc énergie C1-21 C1-24 C1-25 C1-26

(MANTAUX Guéroc, BERNYER Quentin)

Référence de conception : CCPT08

Exigences client vérifiées : EXIG_RCPT_ENERGIE C1-22

La circuit est alimenté par une batterie LIPO 2s qui alimente en 7.4V.

Figure : Calcul de la consommation totale.

accus (Ah), @80% capa.	autonomie récepteur (min):	prix		
0,35	6,24	5,58€		
0,5	8,92	6,24€		
1	17,84	7,08€		
éléments :	courant consommé (A):		total courant (A)	2,69
Moteur brushless @50%	2,5			
ATMEGA328P	0,012			
Led bleue	0,005			
Buzzer	0,002			
controlleur Xreg	0,001			
Récepteur infrarouge	0,0008			
Led Verte	0,01			
servomoteur	0,16			

Pour respecter l'exigence EXIG_RCPT_ENERGIE, nous avons choisi une batterie de 1000 mAh car elle permet une autonomie théorique d'environ 17 minutes. L'autonomie a été calculée en faisant la somme des consommations de chaque composant de la carte, et en faisant le calcul suivant :

(capa_batterie * 0.8) * 60 / somme_consommations

Afin de réguler la tension de la batterie a une utilisable par notre carte (de 7.4V à 5V), nous avons utilisé le contrôleur du moteur brushless, qui possède un régulateur linéaire 5 V intégré afin d'y connecter un contrôleur.

2 . Architecture informatique

2.1. Architecture Émetteur C1-25

Pour développer ce code, nous nous somme aidé des cours d'informatique industrielle. **C1-23**
Vous pouvez retrouver le code final au lien suivant : **C1-35**
https://drive.google.com/drive/u/1/folders/1dwHKSEYppwqNiLJaA5yF_ekSCB2Fljto
dans le dossier Informatique.
Le document se nomme programme émetteur, il suffit de l'ouvrir en double cliquant dessus et de le télécharger en haut à droite. **C1-36**

Code final :

```
#include <arduino.h>
#include "NEC.h"
#define LED_INFRAROUGE 3  // n° de broche de la LED infrarouge
#define POTAR_ROUE A0      // n° de broche du potentiomètre qui contrôle les roues
#define POTAR_HELICE A1    // n° de broche du potentiomètre qui contrôle l'hélice
#define BUZZER 2
#define equipe 0x14
int FinSilence;
unsigned short Roue, Helice;

void setup()
{
  pinMode(LED_INFRAROUGE, OUTPUT);
  pinMode(POTAR_ROUE, INPUT);
  pinMode(POTAR_HELICE, INPUT);
  pinMode(BUZZER, INPUT);

}

void loop()
{
  static unsigned char LastDonneeNEC = 0;
  static unsigned char LastAdresseNEC = 0;
  unsigned char AdresseNEC;

  FinSilence = millis() + 225;
  do
  {
    PotRoue(Roue, Helice);
    int ButBuzzer = AcquerirBuzzer();
    unsigned char DonneeNEC(Roue, Helice);
    AdresseNEC = FAdresseNEC(ButBuzzer, equipe);
  }
  while ((DonneeNEC == LastDonneeNEC) && (LastAdresseNEC == AdresseNEC) && (millis() < FinSilence));
  GenererTrameNEC(LED_INFRAROUGE, AdresseNEC, DonneeNEC);  // Action Roue et Moteur
  LastDonneeNEC = DonneeNEC;  // Traitement Roue et Moteur
  LastAdresseNEC = AdresseNEC;
}

unsigned short PotRoue(unsigned short Roue, unsigned short Helice)
{
  Roue = analogRead(POTAR_ROUE);  //la valeur du potentiometre roue
```

```
  Helice = analogRead(POTAR_HELICE);
}

unsigned short AcquerirBuzzer()
{
  if (digitalRead(BUZZER) == LOW)
  {
    return 1;
  }
  else
  {
    return 0;
  }
}

unsigned char DonneeNEC(unsigned short Roue, unsigned short Helice)
{
  unsigned char NECroue = map(Roue, 0, 1024, 0, 14); //map
  unsigned char NEChelice = map(Helice, 0, 1024, 0, 15); //map
  unsigned char OctetNEC = NECroue;
  OctetNEC << 4;
  OctetNEC | NEChelice;

  return NEChelice;
}

unsigned char FAdresseNEC(int ButBuzzer, int Equipe)
{
  if (ButBuzzer == 0)
  {
    return Equipe;
  }
  else {
    return (0x80 | Equipe);
  }
}
```

2.1.1. Bloc acquisition C1-21 C1-25 C1-26

(MANTAUX Guéroc, BERNYER Quentin)

Référence de conception : CCPT09

> **Exigences client vérifiées :** EXIG_EMTT_TRAITEMENT, EXIG_EMTT_RETENTISEMENT, EXIG_EMTT_IHM, **C1-22**

Pour répondre à l'exigence EXIG_EMTT_IHM qui consiste à la lecture des potentiomètres nous utiliserons la fonction analogRead() et digitalRead() pour la lecture de l'état du bouton poussoir (EXIG_EMTT_RETENTISEMENT)buzzer.

```
unsigned short PotRoue(unsigned short Roue, unsigned short Helice) {  //fonction lecture potentiomètre
  Roue = analogRead(POTAR_ROUE);  //la valeur du potentiometre roue
  Helice = analogRead(POTAR_HELICE);  //la valeur du potentiometre helice
}

unsigned short AcquerirBuzzer() { //fonction lecture bouton buzzer
  if (digitalRead(BUZZER) == LOW) {
    return 1;  //Si bouton appuyer = 1
  }
  else {
    return 0;
  }
}
```

Ces données seront traitées dans la partie 1 du bloc traitement.
La fonction PotRoue lit les valeurs des deux potentiomètres et les enregistre dans deux variables Roue et Hélice.
La fonction AcquerirBuzzer lit la valeur du bouton, elle renvoie 1 si le bouton est appuyé ou 0 si le bouton n'est pas appuyé.

2.1.2. Bloc traitement C1-21 C1-25 C1-26

(MANTAUX Guéroc, BERNYER Quentin)

Référence de conception : CCPT10

Exigences client vérifiées : EXIG_EMTT_TRAITEMENT, EXIG_EMTT_RETENTISEMENT C1-22

Dans cette partie nous expliquerons le fonctionnement de la partie traitement du code.

afin de rendre la partie informatique plus simples, nous avons utilisées des bibliothèques pour les trames NEC :

```
#include <arduino.h>
#include "NEC.h"
```

1) traitement données de l'acquisition

les données acquises par le bouton poussoir pour le buzzer et les potentiomètres ne peuvent pas êtres utilisées directement, il faut donc un étage de traitement supplémentaire on utilise donc une fonction nommé DonneeNEC()

```
unsigned char DonneeNEC(unsigned short Roue, unsigned short Helice)
{
  unsigned char NECroue = map(Roue, 0, 1024, 0, 14); //map
  unsigned char NEChelice = map(Helice, 0, 1024, 0, 15); //map
  unsigned char OctetNEC = NECroue;
  OctetNEC << 4;
  OctetNEC | NEChelice;

  return NEChelice;
}
```

=> Ici, nos fonctions map(variable, valeur_min, valeur_max, valeur_voulue_min, valeur_voulue_max) sont utilisées afin de rendre les valeurs lues en 10 bits des potentiomètres sur 4 bits afin de faire rentrer les données des 2 potentiomètres sur 1 octets pour la trame NEC.
=> OctetNEC << 4 permet déplacer nos données du potentiomètres des roues avant sur les 4 bits forts de la trame sur 8 bits.
=> OctetNEC | NEChelice la fonction ou réalise une trame sur 8 bits contenant les données des roues (NECroue) préalablement mit sur les 4 bits de points forts, et les données de l'hélice (NEChelice) sur les 4 bits de points faibles.

```
unsigned char FAdresseNEC(int ButBuzzer, int Equipe)
{
  if (ButBuzzer == 0)
  {
    return Equipe;
  }
  else {
    return (0x80 | Equipe);
  }
}
```

=> ici, la fonction FAdresse est utilisée afin de rendre la valeur du bouton poussoir et l'adresse de l'équipe utilisables pour êtres envoyées sur une trame sur 8 bits.
=> if (ButBuzzer == 0) vérifie si le bouton est enfoncé ou non.
Si il n'est pas appuyé la fonction renvoie seulement l'adresse de l'équipe, Dans le cas contraire la fonction renvoie renvoie l'adresse de l'équipe sur les 4 bits de point faible et l'adresse de mise sous tension du buzzer sur les 4 bits de points forts

2) utilisation des données traitées

```
void loop()
{
  static unsigned char LastDonneeNEC = 0;
  static unsigned char LastAdresseNEC = 0;
  unsigned char AdresseNEC;

  FinSilence = millis() + 225;
  do
  {
    PotRoue(Roue, Helice);
    int ButBuzzer = AcquerirBuzzer();
    unsigned char DonneeNEC(Roue, Helice);
    AdresseNEC = FAdresseNEC(ButBuzzer, equipe);
  }
  while ((DonneeNEC == LastDonneeNEC) && (LastAdresseNEC == AdresseNEC) && (millis() < FinSilence));
  GenererTrameNEC(LED_INFRAROUGE, AdresseNEC, DonneeNEC);
  LastDonneeNEC = DonneeNEC;
  LastAdresseNEC = AdresseNEC;
```

=> ici, nous avons utilisé un do{}while afin de lire les données précédemment traitées. ces données sont lues en boucle jusqu'à ce que :

- une nouvelle donnée sur le potentiomètre des roues ou de l'hélice (DonneeNEC) soit entré
- une nouvelle donnée sur le buzzer (AdresseNEC) soit entré
- 333 milisecondes soient passées (108ms pour une trame + 225ms d'attente)

=> les données traitées seront ensuite envoyées dans la partie action

2.1.3. Bloc action C1-21 C1-25 C1-26

(HORTOS Sullivan, CAZADE Amory)

Référence de conception : CCPT11

Exigences client vérifiées : EXIG_EMTT_TRAITEMENT, EXIG_EMTT_REPETITIVITE C1-22

```
void loop()
{
  static unsigned char LastDonneeNEC = 0;
  static unsigned char LastAdresseNEC = 0;
  unsigned char AdresseNEC;

  FinSilence = millis() + 225;
  do
  {
    PotRoue(Roue, Helice);
    int ButBuzzer = AcquerirBuzzer();
    unsigned char DonneeNEC(Roue, Helice);
    AdresseNEC = FAdresseNEC(ButBuzzer, equipe);
  }
  while ((DonneeNEC == LastDonneeNEC) && (LastAdresseNEC == AdresseNEC) && (millis() < FinSilence));
  GenererTrameNEC(LED_INFRAROUGE, AdresseNEC, DonneeNEC);
  LastDonneeNEC = DonneeNEC;
  LastAdresseNEC = AdresseNEC;
```

La boucle loop permet l'envoie des trames générées dans les fonctions précédentes.

=> Nous définissons 2 variables qui vont servir de mémoire afin de comparer les trame NEC afin de recalculer la trame ou non. Ceci permet de répondre à l'exigence EXIG_EMTT_REPETITIVITE. Si les potentiomètres ne sont pas déplacés alors la même trame est envoyée une seconde fois.
La valeur des potentiomètres sont lus continuellement tant que les potentiomètres ne sont pas déplacés, cela pendant une période de 225 ms. Si ceux-ci sont déplacés alors on sort de la boucle et une nouvelle trame est envoyée.

=> Les variables AdresseNEC et DonneeNEC sont codées sur 8 bits et permettent de créer une trame NEC. La variable FAdresseNEC qui est égale à AdresseNEC et la variable DonneNEC sont expliquées en partie 1) du bloc traitement.

=> On peut alors envoyer une trame NEC à l'aide de la fonction GenererTrameNEC. Celle-ci prends en paramètre la pin de la LED infrarouge, l'AdresseNEC ainsi que la DonneeNEC

Afin de créer la boucle do{}while, nous avons utilisé les TP vu en informatique industrielle. C1-23

2.2 Architecture Récepteur

Référence de conception : CCPT12

(Petitjean Nathan, Pelamatti Paul, Dumas Adrien)

Exigences client vérifiées : EXIG_RCPT_TRAITEMENT, EXIG_RCPT_SECURITE, EXIG_RCPT_RETENTISSEMENT, EXIG_RCPT_MOTEUR, EXIG_RCPT_ROUE, EXIG_RC PT_CONNEXION, EXIG_RCPT_KLAXON **C1-22**

- Pour développer ce code, nous nous somme aidé des cours d'informatique industrielle. et du programme fournis en aide du projet. **C1-23**

Vous pouvez retrouver le code final au lien suivant : **C1-35**
https://drive.google.com/drive/folders/1J9mzxmQzRHtoXKFFdBdtHa_Wvb0FCF2b?usp=sharing
Ce document se nomme programme émetteur, il suffit de l'ouvrir en double cliquant dessus et de le télécharger en haut à droite. **C1-36**

```
// inclusion des fichiers header des bibliothèques de fonctions Arduino
#include <arduino.h>
#include <Servo.h>
#include "NEC.h"


// definition des constantes du programme
#define TRAME_INFRAROUGE_Pin        5   // n° de broche du recepteur de trame infrarouge
#define LED_VERTE_Pin               2   // n° de broche de la LED verte (Marche/arrêt)
#define KLAXON_Pin                  4   // n° de broche du buzzer (klaxon)
#define LED_BLEUE_Pin               7   // n° de broche de la LED BLEUE (connexion)
#define PWM_ROUE_Pin                5   // n° de broche du signal de pilotage du servomoteur de roue
#define PWM_MOTEUR_Pin              6   // n° de broche du signal de pilotage du moteur brushless
#define AdresseNEC                0x14    // Adresse d'équipe pour le protocole NEC
```

Assignation des pins

```
// declaration des variables globales du programme
Servo ServomoteurRoue;
Servo ServomoteurMoteur;
```

déclaration des variables de vitesse (Moteur) et direction (Roue)

```
// declaration des fonctions du programme
unsigned char CalculerAngleMoteur(unsigned char Donnee);          // fonction de traitement
unsigned char CalculerAngleRoue(unsigned char Donnee);            // fonction de traitement
unsigned char CalculerEtatKlaxon(unsigned char Adresse);          // fonction de traitement
void PiloterServomoteurMoteur(unsigned char Angle);               // fonction d'action
void PiloterServomoteurRoue(unsigned char Angle);                 // fonction d'action
void AllumerDiodeVerte(void);                                     // fonction d'action
void AllumerDiodeBleue(void);                                     // fonction d'action
void EteindreDiodeBleue(void);                                    // fonction d'action
void AllumerKlaxon(void);                                         // fonction d'action
void EteindreKlaxon(void);                                        // fonction d'action
```

Déclarations des fonctions du programme

```
// definition des fonctions du programme
unsigned char CalculerAngleMoteur(unsigned char Donnee)           // CalculerAngleMoteur : fonction de traitement
{
  switch(Donnee >> 4)
  {
  case  0 : return   0;
  case  1 : return  26;
  case  2 : return  37;
  case  3 : return  48;
  case  4 : return  59;
  case  5 : return  70;
  case  6 : return  81;
  case  7 : return  92;
  case  8 : return 103;
  case  9 : return 114;
  case 10 : return 125;
  case 11 : return 136;
  case 12 : return 147;
  case 13 : return 158;
  case 14 : return 169;
  case 15 : return 180;
  }
}
```

Fonction de traitement Moteur

```
unsigned char CalculerAngleRoue(unsigned char Donnee)             // CalculerAngleRoue : fonction de traitement
{
  switch (Donnee & 0x0F)
  {
  case  0 : return  0;
  case  1 : return  22;
  case  2 : return  41;
  case  3 : return  57;
  case  4 : return  70;
  case  5 : return  80;
  case  6 : return  86;
  case  7 : return  90;
  case  8 : return  94;
  case  9 : return 100;
  case 10 : return 110;
  case 11 : return 123;
  case 12 : return 139;
  case 13 : return 158;
  case 14 : return 180;
  default : return  106;
  }
}
```

Fonction de traitement Roue

```
unsigned char CalculerEtatKlaxon(unsigned char Adresse)            // CalculerEtatKlaxon : fonction de traitement
{
  if ((Adresse & 0x80) == 0)                                       /* Utilisation du bit de poids
      fort de l'adresse NEC pour transmettre l'information de klaxon*/
  return 0;
  else
  return 1;
}
```

Fonction de traitement klaxon

```
void PiloterServomoteurMoteur(unsigned char Angle)                // PiloterServomoteurMoteur : fonction d'action
{
  ServomoteurMoteur.write(Angle);
}
```

Fonction action Moteur

```
void PiloterServomoteurRoue(unsigned char Angle)                  // PiloterServomoteurRoue : fonction d'action
{
  ServomoteurRoue.write(Angle);
}
```

Fonction action Roue

```
void AllumerDiodeVerte(void)                                      // AllumerDiodeVerte : fonction d'action
{
  digitalWrite(LED_VERTE_Pin, HIGH);
}
```

Fonction allumage led verte

```
void AllumerDiodeBleue(void)                                      // AllumerDiodeBleue : fonction d'action
{
  digitalWrite(LED_BLEUE_Pin, HIGH);
}

void EteindreDiodeBleue(void)                                     // EteindreDiodeBleue : fonction d'action
{
  digitalWrite(LED_BLEUE_Pin, LOW);
}
```

Fonction allumage et éteinte led bleu

```
void AllumerKlaxon(void)                                          // AllumerKlaxon :fonction d'action
{
  tone(KLAXON_Pin, 4000);
}

void EteindreKlaxon(void)                                         // EteindreKlaxon : fonction d'action
{
  noTone(KLAXON_Pin);
}
```

Fonction allumage et éteinte klaxon

4.1 Conclusion de la conception détaillée du produit

La conception a permis d'élaborer le schéma électrique du produit et de dimensionner tous les composants ce qui donne accès au dossier de fabrication. Les valeurs des composants déterminées

dans cette étude sont données dans le tableau ci-dessous. Ces valeurs feront l'objet d'une vérification en simulation et/ou prototypage rapide (voir paragraphe 4) et en essai (voir DDV).

Schéma électrique récepteur

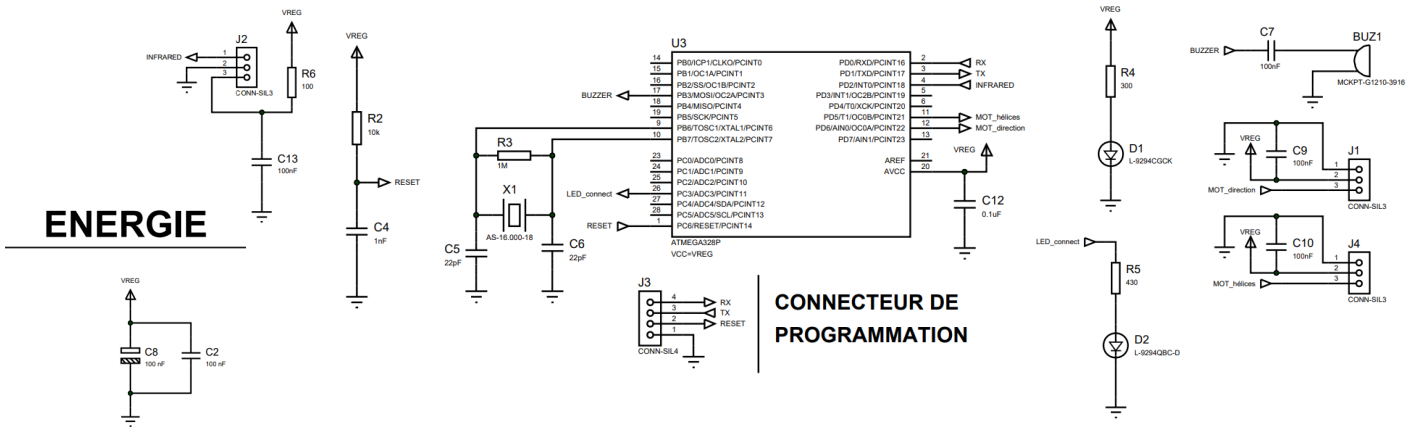
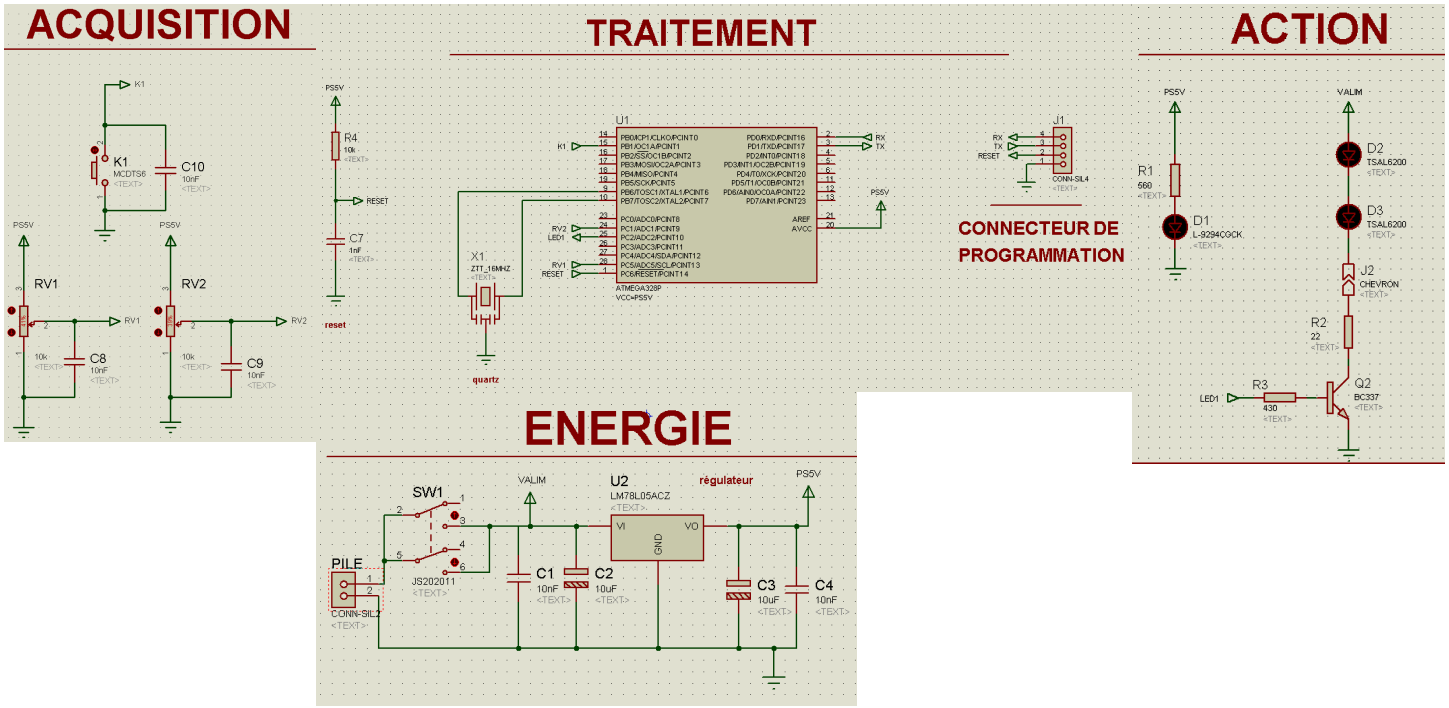


Schéma électrique émetteur



Dérisquage des solutions techniques retenues

Ce chapitre détaille les activités de dérisquage des solutions techniques retenues : simulation et/ou prototypage rapide. Il constitue une preuve partielle de la conformité du produit. Chaque paragraphe de l'étude fait donc clairement référence aux exigences client issues du [CDC].

Il permet également de confirmer les résultats théoriques effectués aux paragraphes 2 et 3 en vérifiant le fonctionnement à travers des simulations et/ou prototypages rapides. Pour chaque simulation et/ou prototypage rapide est renseigné le protocole de mise en œuvre. Les résultats des simulations et/ou prototypages rapides sont confrontés aux résultats de l'étude théorique.

1.1. Prototypage émetteur traitement

(CAZADE Amory, BERNYER Quentin)

Référence de la simulation : SIM1

Exigences client vérifiées : EXIG_EMTT_TRAITEMENT

But de la simulation : Vérifier la création de tram NEC

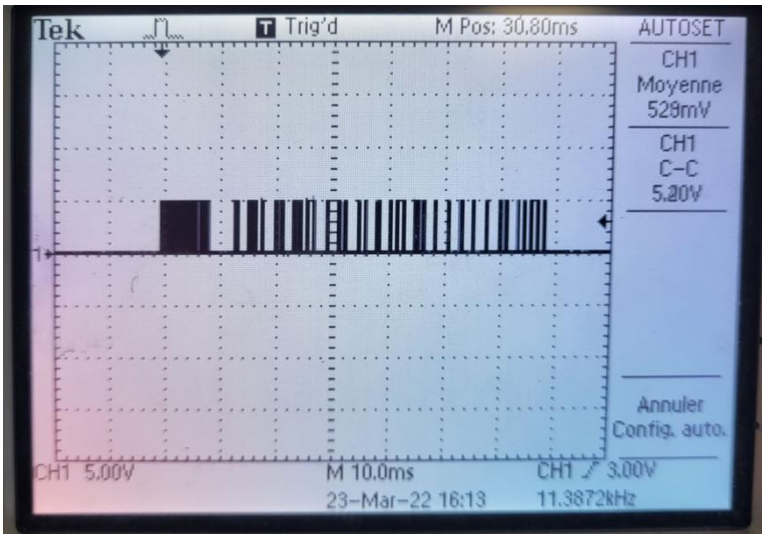
Procédure de simulation : Lecture du signal émis par le microcontrôleur sur un oscilloscope :

Pour réaliser le prototypage nous avons utilisé une carte arduino uno qui a permis de simuler le comportement du micro contrôleur. Nous avons réalisé le schéma électrique traitement et avons simulé l'alimentation a grâce à une alim de table. Nous avons donc pu visualiser l'information envoyée grâce à un oscilloscope.

Résultats attendus : Tram NEC

Grandeur	Valeur attendue	Tolérance
Période	333 ms	+/- 10%

Résultats obtenus :



Grandeur	Valeur mesurée	Conf/Non conf.
Période	325ms	Conforme.

Statut de la simulation :

La simulation est conforme aux attentes. Chaque partie de la tram NEC (entête, adresse et données) est cohérente.

Problèmes rencontrés :

Nous n'avons constaté aucun problème lors des tests.

1.2. Prototypage émetteur action

(MANTAUX Guéroc, BERNYER Quentin)

Référence de la simulation : SIM2

Exigences client vérifiées : EXIG_EMTT_PUISSANCE

But de la simulation : Vérifier l'envoie d'un signal infrarouge.

Procédure de simulation :

Lecture du signal infrarouge émis par les leds grâce à une caméra :

Un signal infrarouge ne peut être observé avec l'œil humain mais une caméra comme celle d'un téléphone peut renvoyer l'image de ce signal. Nous avons donc pu vérifier l'émission du signal grâce à un smartphone.

Résultats attendus :

Grandeur	Valeur attendue	Tolérance
Signal visible par la caméra	OUI	X

Résultats obtenus :

Grandeur	Valeur mesurée	Conf/Non conf.
Signal visible par la caméra	OUI	Conforme

Statut de la simulation :

Nous pouvons observer les leds infrarouge s'allumer sur l'écran du smartphone cela signifie que l'action "envoie de la tram NEC" est bien réalisée.

Problèmes rencontrés :

Nous n'avons constaté aucun problème lors des tests.

1.3. Prototypage émetteur action

Référence de la simulation : SIM4

(Petitjean Nathan, Adrien Dumas, Paul pelamatti)

Exigences client vérifiées : EXIG_EMTT_PUISSANCE

But de la simulation : tester la l'emission pour 10m.

Procédure de simulation :

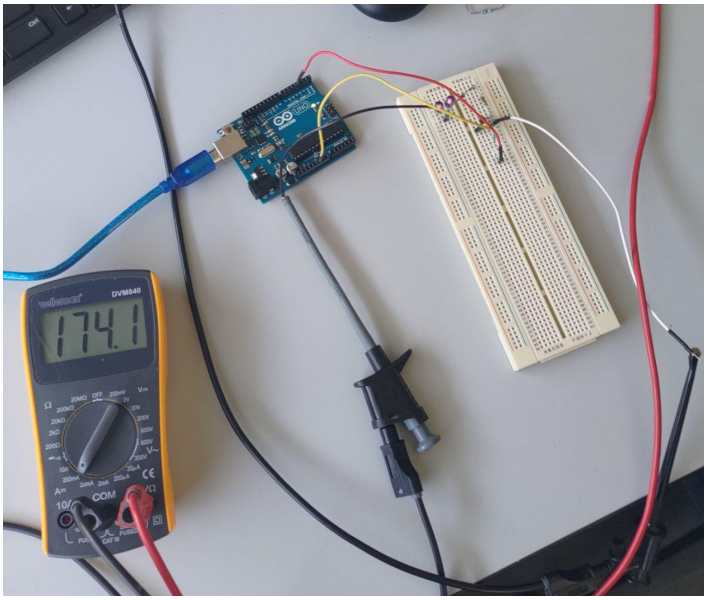
nous mettons le système de réception et d'émission à 10m puis observons si la réception s'effectue

Résultats attendus :

nous devons recevoir les trame a une distance de 10m

Grandeur	Valeur attendue	Tolérance
distance de réception	10 m	+ 10%

Résultats obtenus :



Grandeur	Valeur mesurée	Conf/Non conf.
distance de réception	10m	Conforme.

Statut de la simulation :

La réception à 10m s'effectue correctement.

Problèmes rencontrés :

Nous n'avons constaté aucun problème lors des tests.

1.4. Prototypage Récepteur acquisition

Référence de la simulation : SIM5

(Petitjean Nathan, Paul pelamatti)

Exigences client vérifiées : EXIG_RCPT_CAPTEUR

But de la simulation : Recevoir une trame NEC depuis un émetteur NEC

Procédure de simulation :

Nous émettons une Trame NEC via des LED infrarouge, et nous essayons de recevoir la trame

Résultats attendus : Pour atteindre les résultats attendus, avec le test, nous devons obtenir un signal carré de 5 V.

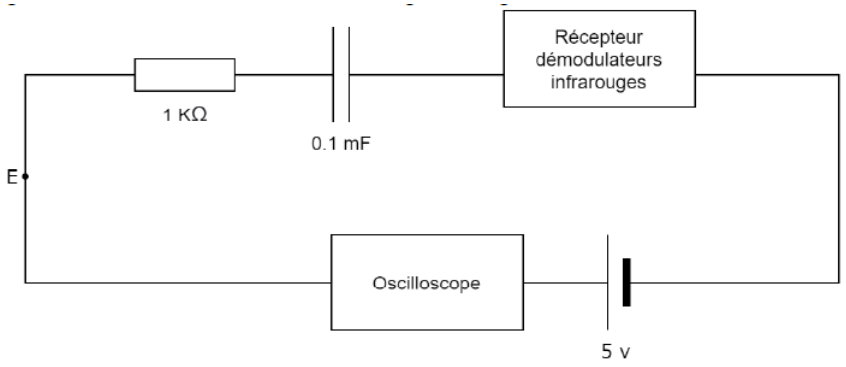


Figure : Schéma du test du récepteur infrarouge

Résultats obtenus : Nous avons obtenu un signal carré qui est constant de 4,9V lors d'une utilisation d'une télécommande infrarouge.

Statut De la simulation : Conforme au cahier des charges

Problèmes rencontrés : Le principal problème rencontré est que le modèle utilisé d'oscilloscope ne permet pas de récupérer l'image du signal. Nous avons également essayé de simuler via un Arduino mais cela n'a rien donné de concluant.

1.5. Conclusion de la simulation / prototypage rapide du produit

La simulation montre la conformité de chaque bloc fonctionnel vis-à-vis des exigences du cahier des charges.

IV Conclusion de la conception du produit

La simulation de chaque partie : Acquisition, Traitement de l'information , Action et Énergie : la totalité des simulations ont révélé une conformité du produit vis-à-vis du cahier des charges. La totalité du fonctionnement est correcte, le processus de fabrication peut désormais commencer.

Matrice de conformité du produit

Ce chapitre synthétise par l'intermédiaire d'un tableau la conformité du produit développé par rapport aux exigences issues du Cahier des Charges.

	Exigence	Méthodes Vérification	Eléments vérifiant l'exigence	Statut
Emetteur	EXIG_EMTT_DIMENSIONS	Conception Conception/Fab. Vérification	PRC01, FAB03, FAB04, FAB05,	Conf.
Emetteur	EXIG_EMTT_LOGO	Conception Conception/Fab. Vérification	PRC01	Conf.
Emetteur	EXIG_EMTT_ENERGIE	Conception Vérification	PRC03, PRC08, CCPT04, FAB01	Conf.
Emetteur	EXIG_EMTT_INTERRUPTEUR	Conception Conception/Fab. Vérification	PRC01, PRC03, PRC09, FAB01	Conf.
Emetteur	EXIG_EMTT_IHM	Conception Conception Simulation Con ception/Fab. Vérification	PRC01, PRC03, PRC04 PRC15, CCPT01, CCPT09, FAB01	Conf.
Emetteur	EXIG_EMTT_KLAXON	Conception Conception Simulation Conception/Fab.	PRC01, PRC03, PRC04, PRC15, CCPT01,	Conf.

		Vérification	FAB01	
Emetteur	EXIG_EMTT_TRAITEMENT	Conception Conception/Fab. Vérification	PRC03, PRC05, PRC16, CCPT02, CCPT09, CCPT10, FAB01, FAB01,	Conf.
Emetteur	EXIG_EMTT_REPETITIVITE	Conception Vérification	PRC03, PRC05 , PRC06, PRC16 , CCPT02, CCPT11	Conf.
Emetteur	EXIG_EMTT_RETENTISSEMENT	Conception Conception/Fab. Vérification	PRC03, PRC05, PRC16, CCPT02, CCPT09, CCPT10, FAB01	Conf.
Emetteur	EXIG_EMTT_PUISSANCE	Conception Conception/Fab. Vérification	PRC03, PRC07, PRC17, CCPT03, SIM4, FAB01	Conf.
Emetteur	EXIG_EMTT_INDICATEUR	Conception Conception/Fab.	PRC01, PRC03, PRC07, PRC17, CCPT03, CCPT04, FAB01	Conf.
tout le projet	EXIG_COUT	Conception/Fab	FAB02 FAB07	conf

Eléments concernés	Exigence	Méthodes Vérification	Eléments vérifiant l'exigence	Statut
Récepteur	EXIG_RCPT_DIMENSIONS	par inspection documentaire par mesure	PRC02	Conforme
Récepteur	EXIG_RCPT_LOGO	par observation visuelle	PRC02	Conforme
Récepteur	EXIG_RCPT_ENERGIE	par analyse et calculs par essai	PRC14, CCPT08	Conforme
Récepteur	EXIG_RCPT_CAPTEUR	par analyse et calculs par essai	PRC10, PRC11, CCPT05, CCPT06	Conforme
Récepteur	EXIG_RCPT_TRAITEMENT	par analyse et calculs par essai	PRC10, PRC12, CCPT06,	Conforme
Récepteur	EXIG_RCPT_SECURITE	par analyse et calculs par essai	PRC10, PRC12, CCPT06	Conforme
Récepteur	EXIG_RCPT_RETENTISSEMENT	par analyse et calculs par essai	PRC10, PRC12, CCPT06	Conforme
Récepteur	EXIG_RCPT_MOTEUR	par analyse et calculs par essai	PRC10, PRC13, CCPT06, CCPT07	Conforme
Récepteur	EXIG_RCPT_ROUE	par analyse et calculs par essai	PRC10, PRC13, CCPT06, CCPT07	Conforme
Récepteur	EXIG_RCPT_INDICATEUR	par analyse et calculs par essai	PRC10, PRC13, CCPT06, CCPT07	Conforme
Récepteur	EXIG_RCPT_CONNEXION	par analyse et calculs par essai	PRC10, PRC13, CCPT06, CCPT07	Conforme
Récepteur	EXIG_RCPT_KLAXON	par analyse et calculs par essai	PRC10, PRC13, CCPT06, CCPT07	Conforme