Dossier De Conception (DDC)

du projet

Kart À Hélice (KAH)

Responsabilité documentaire

Action	NOM Prénom	Fonction	Date	Signature
Rédigé par	Mathéo GRILLET, Mathis BROUSSE, Clément CACHO, Nicolas TISSOT, Thomas GIBELIN, Ylhan DELANNOY, Maxence CORDEAU	Technicien	14/02/2024	
Approuvé par	<Chef de projet> (IUT GEII Bdx)	Chef de projet	JJ/MM/AAAA	
Approuvé par	<Client> (Toy Corporation)	Client	JJ/MM/AAAA	

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	14/02/2024	Première publication

Documents de références

Sigle	Référence	Titre	Rév.	Origine
[CDC]	KAH_CDC	Cahier des charges	1	Baby Corporation

Table des matières

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.

Conception préliminaire du produit

Compétences GEII : C1-4

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.

2.1 Mécanique

2.1.1 Mécanique - Émetteur

Référence du paragraphe : CPR_EMTT_ARCHI_MECA

Rédacteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Relecteur : Delannoy Ylhan Cordeau Maxence Tissot Nicolas Gibelin Thomas

Compétences GEII : C1-9

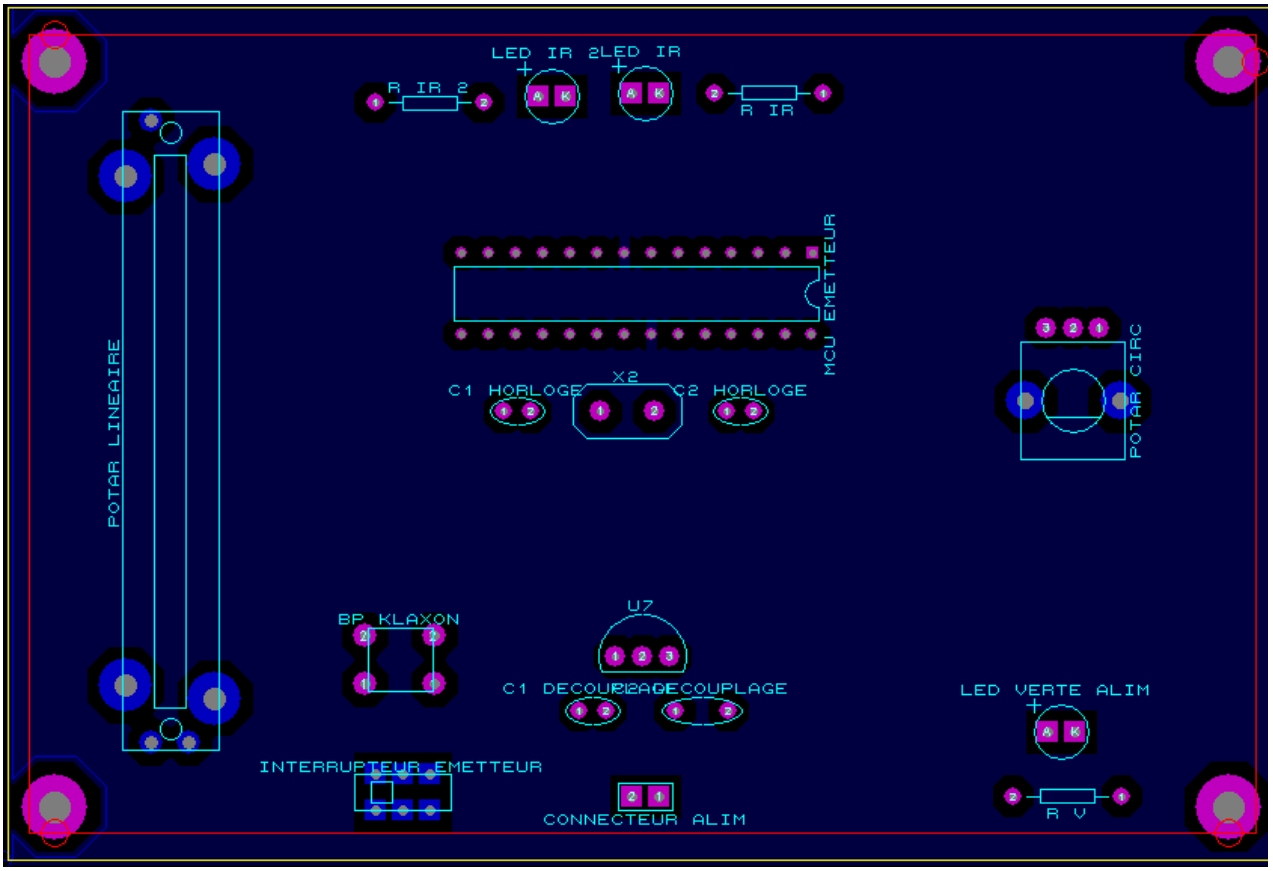


Figure 1 : plan mécanique de l'émetteur

Référence du paragraphe : CPR_EMTT_DIMENSIONS

Rédacteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Relecteur : Delannoy Ylhan Cordeau Maxence Tissot Nicolas Gibelin Thomas

Exigences client vérifiées par pré-conception : EXIG_EMTT_DIMENSIONS

Compétences GEII : C1-10

Nous avons dimensionné la carte électronique sur le logiciel ARES en fonction des exigences précisées par le CDC : 120mm (-/+1mm) en largeur et 80mm (-/+1mm) en longueur, des trous de 3 mm (-/+0,5mm) situés dans chaque coin et les centres de ces trous sont placés à 5mm (-/+0,5mm) des bords du circuit imprimé. Nous avons placé les composants déterminés sur le diagramme architectural (voir figure 3). Nous voyons que nous pouvons placer tous les composants tout en respectant les dimensions données.

Nous avons placée les différents composants en raisonnant comme ci-dessous:

- Les potentiomètres ont été placés en s'inspirant de l'ergonomie des émetteurs concurrents. (rectiligne pour la vitesse à gauche et rotatif pour la direction à droite)
- Le bouton poussoir du klaxon a été placé à gauche étant donné que la main gauche pilote la vitesse on préfèrera donc lâcher cette commande à celle liée à la direction.
- Les LED Infrarouges ont été positionnées à l'avant de la carte afin de faciliter la communication avec le récepteur.

Référence du paragraphe : CPR_EMTT_LOGO

Rédacteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Relecteur : Delannoy Ylhan Cordeau Maxence Tissot Nicolas Gibelin Thomas

Exigences client vérifiées par pré-conception : EXIG_EMTT_LOGO

Compétences GEII : C1-10

Pour cette exigence, nous allons mettre à l'arrière du support de la carte notre logo que nous désignerons dans la conception détaillée. Il contiendra le logo GEII et le numéro de notre équipe.

2.1.2 Mécanique - Récepteur

Référence du paragraphe : CPR_RCPT_ARCHI_MECA

Rédacteur : Delannoy Ylhan Cordeau Maxence Tissot Nicolas Gibelin Thomas

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Compétences GEII : C1-9

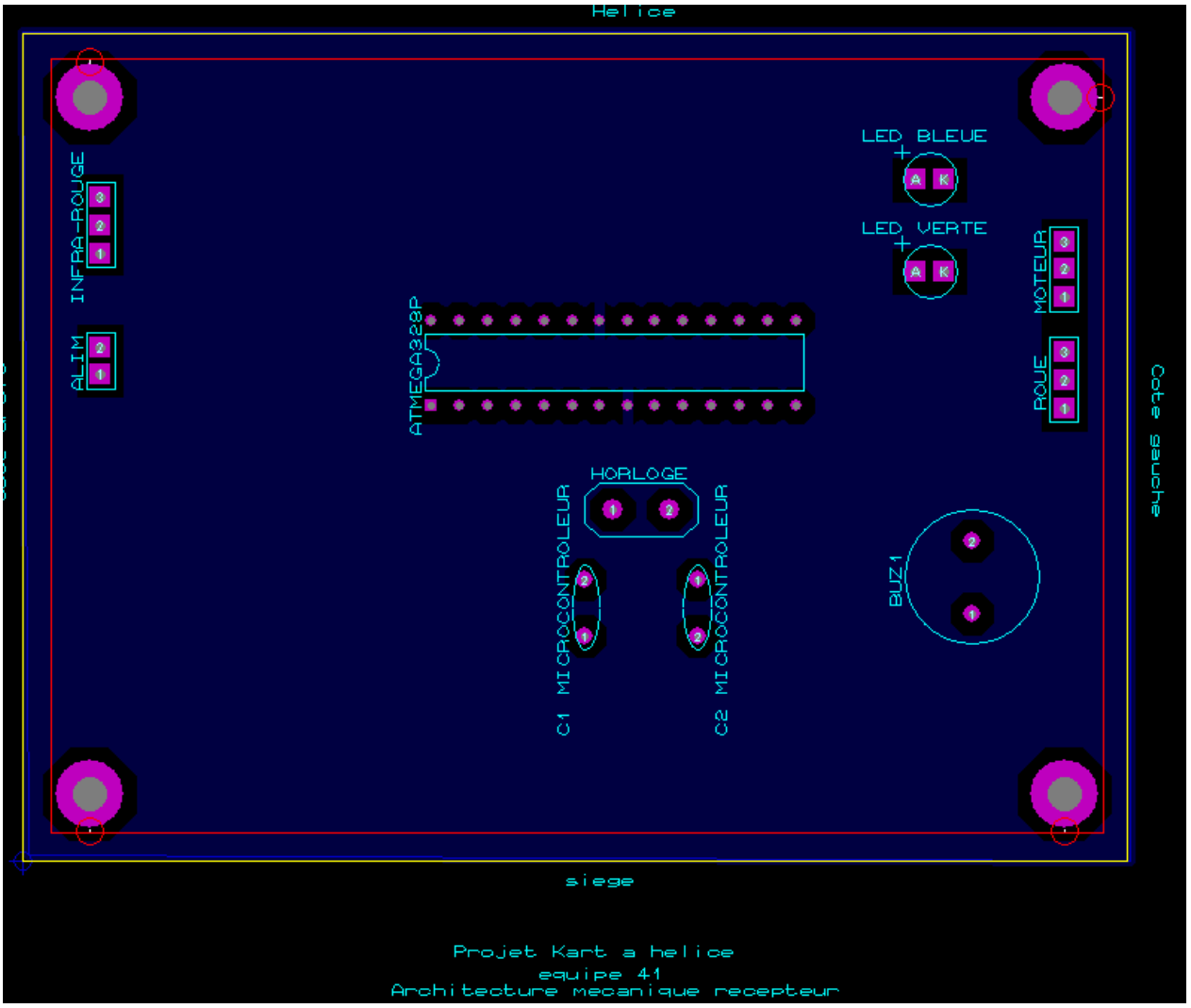


Figure 2 : plan mécanique du récepteur

Référence du paragraphe : CPR_RCPT_DIMENSIONS

Rédacteur : Delannoy Ylhan Cordeau Maxence Tissot Nicolas Gibelin Thomas

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Exigences client vérifiées par pré-conception : EXIG_RCPT_DIMENSIONS

Compétences GEII : C1-10

Le circuit imprimé du récepteur a des dimensions de 100mm en largeur et 75 mm en longueur et comporte des trous de fixation de 3 mm situés dans chaque coin pour fixer celui-ci au reste de la mécanique du kart. Le centre de ces trous sont placés à 6mm des bords du circuit imprimé.

Référence du paragraphe : CPR_RCPT_LOGO

Rédacteur : Delannoy Ylhan Cordeau Maxence Tissot Nicolas Gibelin Thomas

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Exigences client vérifiées par pré-conception : EXIG_RCPT_LOGO

Compétences GEII : C1-10

Le kart comporte sur le carter de la colonne de direction le numéro de l'équipe (police : arial ; taille : 40 ; effet : gras), le nom de l'équipe et le logo du département GEII de l'IUT de Bordeaux.

2.2 Électronique

2.2.1 Électronique - Émetteur

Référence du paragraphe : CPR_EMTT_ARCHI_ELEC

Rédacteur : Mathis BROUSSE, Clément CACHO et Mathéo GRILLET

Relecteur : Mathis BROUSSE, Clément CACHO et Mathéo GRILLET

Compétences GEII : C1-3, C1-9, C1-11

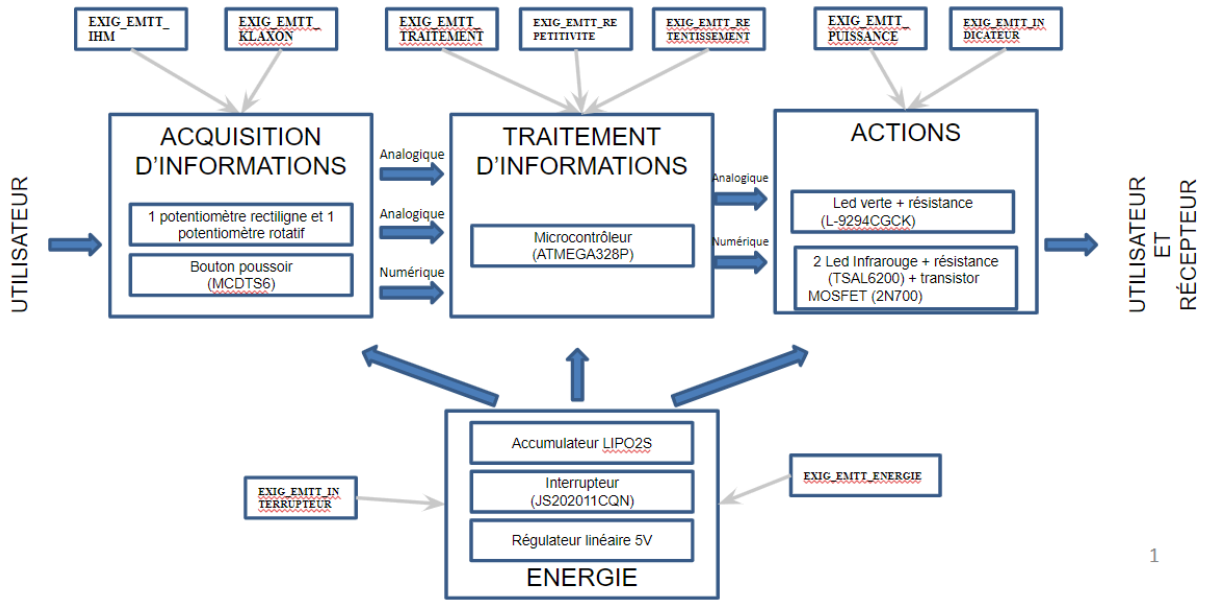


Figure 3 : synoptique architecture électronique de l'émetteur

Référence du paragraphe : CPR_EMTT_IHM

Rédacteur : Grillet Mathéo

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées par pré-conception : EXIG_EMTT_IHM

Compétences GEII : C1-10, C1-11

L'émetteur comporte 2 interfaces homme-machine qui permettent de contrôler respectivement la puissance du moteur et la direction des roues avant du kart. Pour ce qui est de la gestion de la vitesse nous allons utiliser un potentiomètre rectiligne pour des raisons ergonomiques (voir branchement Figure 4, et photo Figure 5). Pour la direction nous avons choisi un potentiomètre rotatif pour des raisons ergonomiques également (voir branchement Figure 6, et photo Figure 7). Nous nous sommes inspirées de la disposition des contrôleurs réalisés par nos concurrents.

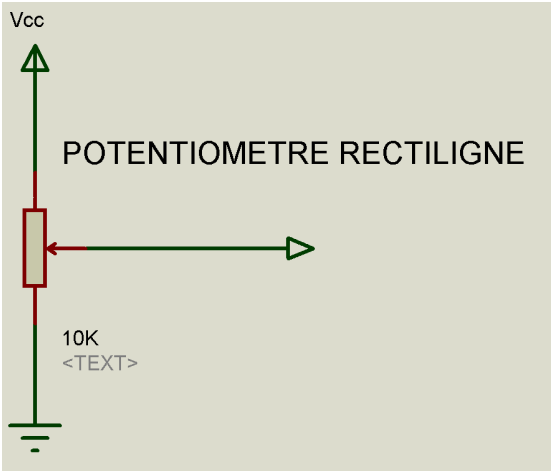


Figure 4: schéma électrique préliminaire du potentiomètre rectiligne

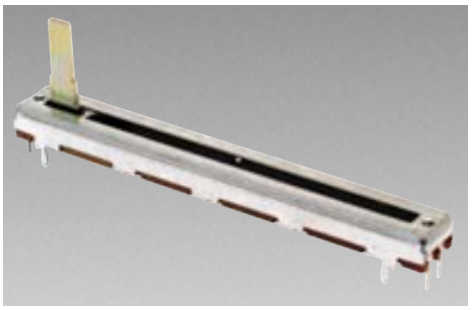


Figure 5: Photo potentiomètre rectiligne

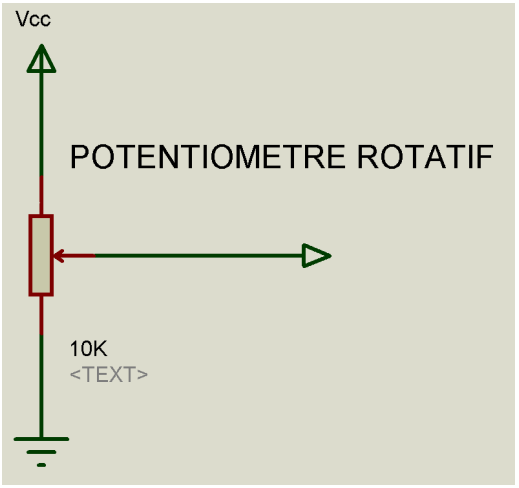


Figure 6: schéma électrique préliminaire du potentiomètre rotatif

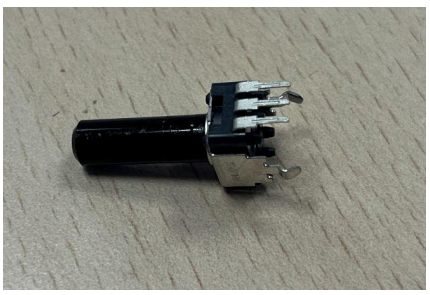


Figure 7: photo potentiomètre rotatif

Référence du paragraphe : CPR_EMTT_KLAXON

Rédacteur : Grillet Mathéo

Relecteur : Mathis BROUSSE, Clément CACHO

Exigences client vérifiées par pré-conception : EXIG_EMTT_KLAXON

Compétences GEII : C1-10, C1-11

L'émetteur comporte un bouton poussoir sur lequel l'utilisateur peut appuyer pour indiquer qu'il souhaite faire retentir le klaxon du kart. Le bouton poussoir sera connecté en pull up c'est-à-dire en logique inversée mais avec l'utilisation de la résistance interne du MCU afin de réduire le coût et l'espace (pas de résistance supplémentaire). (voir schéma de branchement Figure 8)

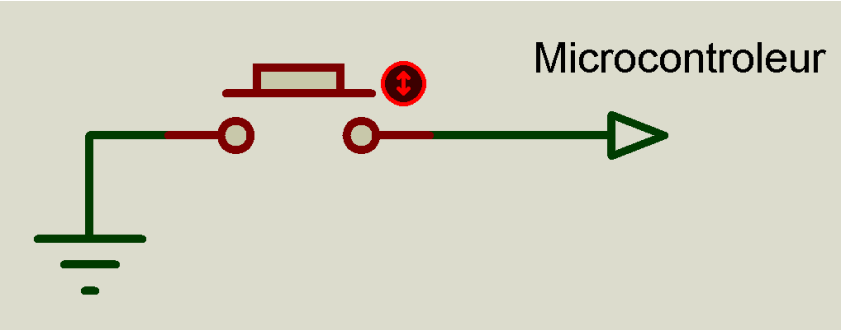


Figure 8: schéma électrique préliminaire du bouton poussoir

Référence du paragraphe : CPR_EMTT_TRAITEMENT

Rédacteur : CACHO Clément et BROUSSE Mathis

Relecteur : GRILLET Mathéo

Exigences client vérifiées par pré-conception : EXIG_EMTT_TRAITEMENT

Compétences GEII : C1-10, C1-11

Afin d'assurer le traitement de l'information sur la partie émetteur nous allons utiliser un microcontrôleur (MCU). Le but premier est de lire la valeur d'entrée liée à la partie acquisition, en l'occurrence les potentiomètres qui vont correspondre à la commande de puissance et d'orientation des roues. Nous avons donc étudié les datasheets des microcontrôleurs proposés afin de savoir lequel choisir. Dans notre cas, nous avons étudié le fonctionnement de l'ADC (Analog to Digital Converter), qui a pour rôle de convertir une valeur analogique en valeur numérique. Ce composant nous permettra de lire la valeur donnée par les potentiomètres et ainsi nous retrouverons la position de celui-ci à l'aide d'un programme informatique. L'exigence traitement mentionne l'utilisation de

8 bits (4 bits pour la puissance moteur et 4 bits pour la direction des roues). Les deux MCU proposés (ATMEGA328P ET ATSAMD21) utilisent respectivement 8 bits et 32 bits, ils sont donc compatibles avec notre utilisation. Or d'après le diagramme d'architecture électronique de l'émetteur nous avons besoin de 2 entrées analogiques, 1 entrée numérique, 1 sortie numérique et 1 sortie analogique. Le MCU ATSAMD21 possède bien plus d'entrées et sorties que le ATMEGA328P qui lui possède suffisamment d'entrées/sorties. (voir Figure 9)

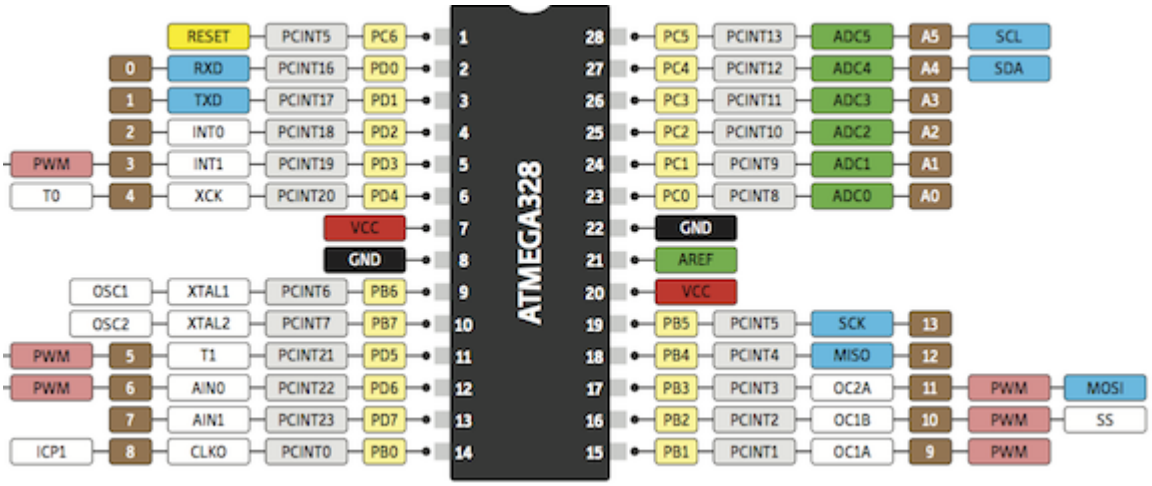


Figure 9 : Entrées sorties de l'ATMEGA328P

Notre choix final est économique car l'ATMEGA328P vaut 2.62€(HT) et l'ATSAMD21 vaut 3.91€(HT), de plus ce dernier consomme bien plus du fait qu'il ait un nombre de broches plus important. Nous choisirons donc l'ATMEGA328P. Nous avons listé les broches disponibles et compatibles avec chaque signal.

Les deux signaux de potentiomètres doivent être branchés sur une broche pouvant recevoir un signal analogique afin de le convertir en signal numérique. On peut donc utiliser les broches 23 à 28 de notre microcontrôleur.

Le bouton poussoir doit être branché sur une broche permettant de recevoir un signal numérique soit toutes les broches sauf alimentation, masse et référence.

La LED verte sera branchée directement à l'alimentation pour des raisons de simplification du montage.

Les deux LED infrarouges doivent être branchées sur une broches capable d'envoyer un signal PWM. On peut alors utiliser les broches 5, 6, 7, 15, 16 et 17 de notre microcontrôleur.

La masse doit être reliée à la broche GND cachée sur le schéma.

La tension d'alimentation doit être reliée soit à la broche 7 soit à la broche 20.

La réinitialisation du MCU doit être branché sur la broche 1.

L'horloge de fonctionnement du MCU doit être branché sur les broches 9 et 10. L'horloge fonctionne à l'aide de 2 condensateurs de découplage placés en parallèle.

Le téléchargement du programme par le biais de l'ordinateur se fera sur la broche 2.

Le signal reset doit être branché sur la broche 1. Selon la datasheet de l'ATMEGA328P, le signal reset se branche à l'aide d'une résistance de 10kΩ et d'un condensateur de 1nF.

Nous avons branché le microcontrôleur de la manière suivante: (voir Figure 10)

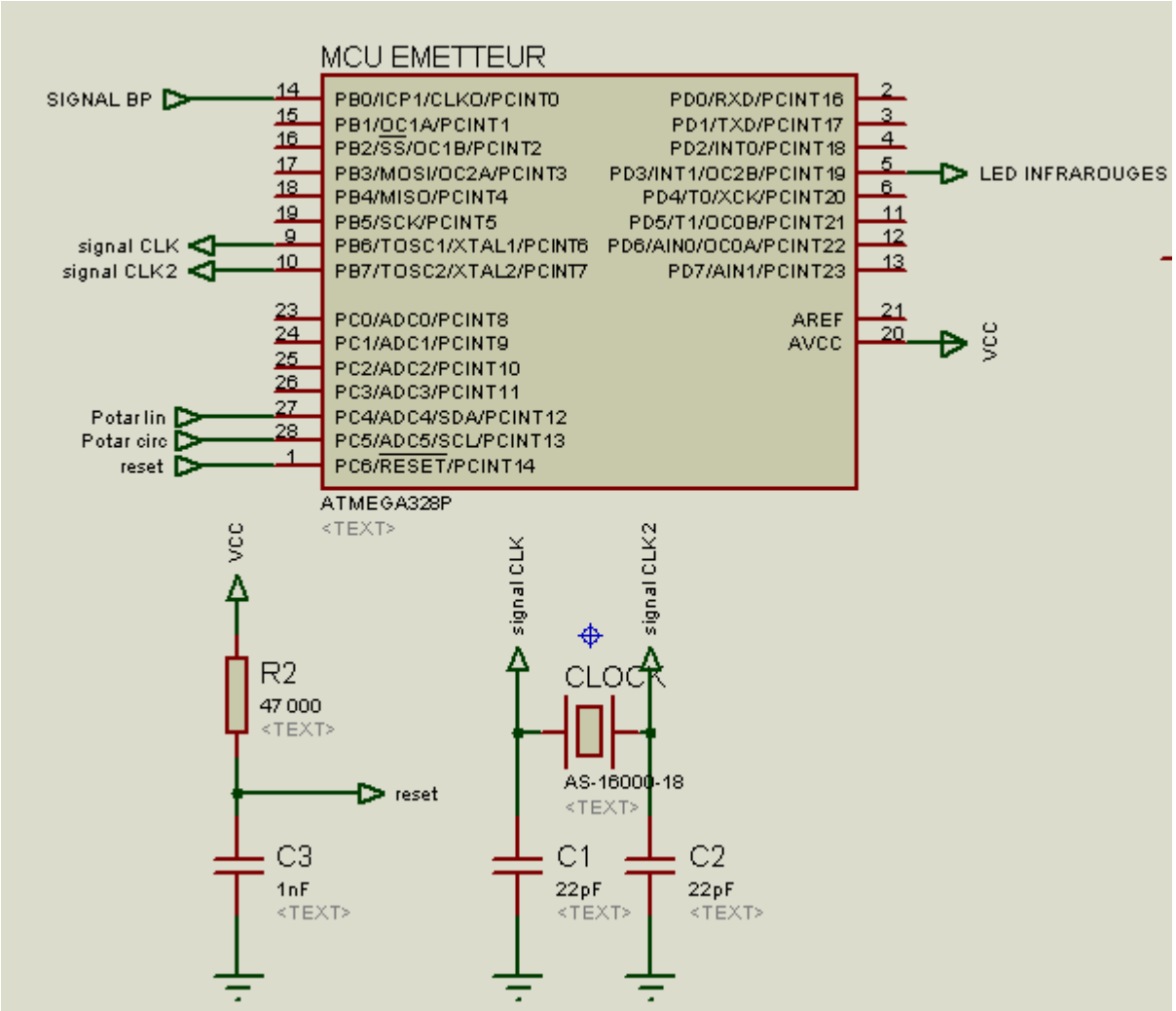


Figure 10 : Branchement du MCU, de l'horloge et du reset

Référence du paragraphe : CPR_EMTT_REPETITIVITE

Rédacteur : CACHO Clément et BROUSSE Mathis

Relecteur : Grillet Mathéo

Exigences client vérifiées par pré-conception :

Compétences GEII : C1-10, C1-11

Cette exigence sera détaillée lors de la partie de conception détaillée. Pour ce faire nous utiliserons le CPU et un Timer en guise de périphérique afin de générer des signaux respectant certaines durées. Lorsque deux trames consécutives sont différentes, elles seront espacées de 108ms +/-10%, tandis que si celles-ci sont similaires, elles seront espacées de 333ms +/-10%.

Référence du paragraphe : CPR_EMTT_RETENTISSEMENT

Rédacteur : CACHO Clément et BROUSSE Mathis

Relecteur : Grillet Mathéo

Exigences client vérifiées par pré-conception : EXIG_EMTT_RETENTISSEMENT

Compétences GEII : C1-10, C1-11

Nous programmerons le retentissement du klaxon dans la partie de conception détaillée. Pour ce faire nous utiliserons le CPU de notre microcontrôleur afin de générer un signal portant l'information du klaxon actif ou inactif.

Référence du paragraphe : CPR_EMTT_PUISSANCE

Rédacteur : GRILLET Mathéo

Relecteur : CACHO Clément et BROUSSE Mathis

Exigences client vérifiées par pré-conception : EXIG_EMTT_PUISSANCE

Compétences GEII : C1-10, C1-11

L'émetteur émet les trames protocolaires à l'aide d'un composant d'émission infrarouge. La puissance crête d'émission infrarouge permet d'assurer à l'émetteur une émission infrarouge décodable par le récepteur à une distance minimum de 10m.

Afin d'obtenir une portée de 10 mètres nous devons utiliser 2 LED traversées par un courant supérieur à 200mA. Par lecture des datasheets, nous relevons qu'un courant compris entre 200 et 300 mA sera adapté à l'utilisation des LED infrarouges pour une communication NEC. On fixera donc une valeur de 250 mA +/- 50 mA.

Le MCU n'est pas capable de fournir 250 mA il faudra donc le connecter avec un transistor MOSFET afin d'amplifier ce courant.

Afin de réduire la consommation et d'utiliser une seule résistance nous allons brancher nos 2 LED en série. (voir branchement Figure 11)

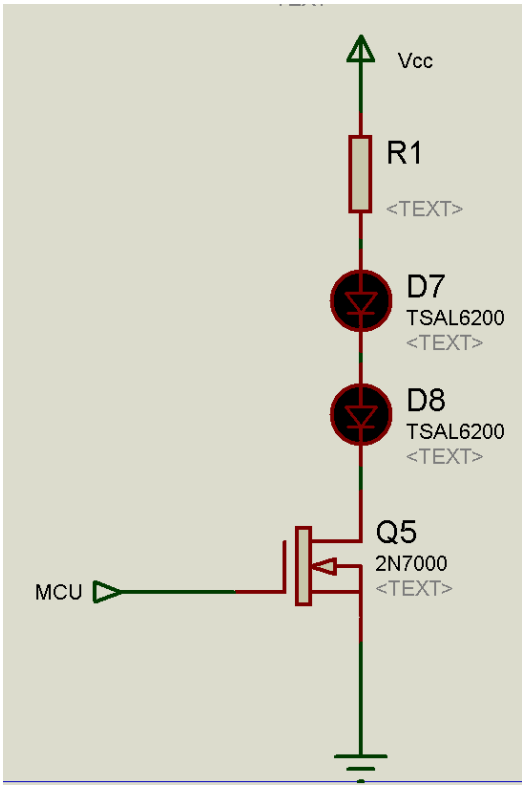


Figure 11: schéma électrique préliminaire des LEDs Infrarouges .

Référence du paragraphe : CPR_EMTT_INDICATEUR

Rédacteur : Grillet Mathéo

Relecteur : CACHO Clément et BROUSSE Mathis

Exigences client vérifiées par pré-conception : EXIG_EMTT_INDICATEUR

Compétences GEII : C1-10, C1-11

L'émetteur du kart comporte un indicateur lumineux (50mcd -/+20%) informant l'utilisateur que l'émetteur est sous tension. L'utilisation d'une LED verte en série avec une résistance semble adaptée. Il est inutile de la connecter au microcontrôleur puisque si la carte est hors tension Vcc sera donc égale à 0V. (voir branchement Figure 12). Afin de respecter le cahier des charges (intensité lumineuse) la tension doit être fixée puisque l'intensité lumineuse est proportionnelle au courant qui traverse la LED. Il faudra une tension supérieure à la tension de seuil, en l'occurrence 2.5V.

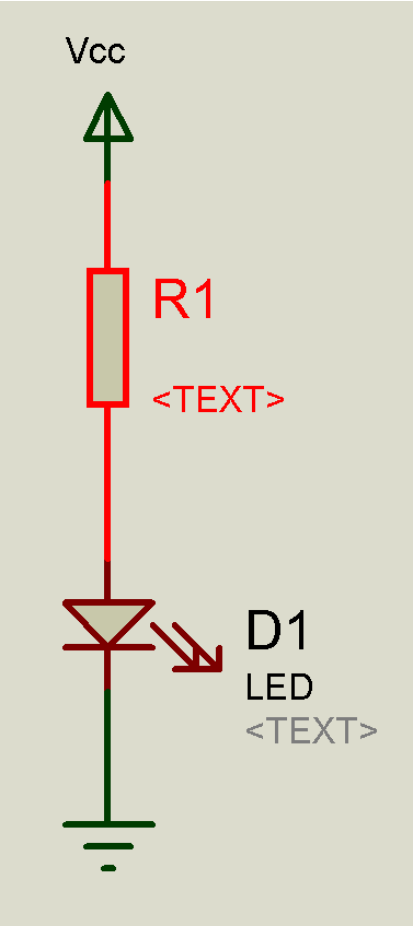


Figure 12: schéma électrique préliminaire de la LED verte

Référence du paragraphe : CPR_EMTT_ENERGIE

Rédacteur : Mathis BROUSSE et Clément CACHO

Relecteur : GRILLET Mathéo

Exigences client vérifiées par pré-conception : EXIG_ENERGIE

Compétences GEII : C1-10, C1-11

Le cahier des charges exige l'utilisation d'un accumulateur Lithium Polymère 2S. Nous utiliserons donc cet accumulateur afin d'assurer l'alimentation de l'émetteur. Cependant nous devons vérifier si les tensions d'alimentations des composants de la carte sont compatibles avec la tension nominale délivrée par l'accumulateur, c'est-à-dire 7,4V. Nous allons donc établir un bilan des tensions d'alimentations (voir tableau suivant).

Nom du composant	Tension d'alimentation (V)
Microcontrôleur ATMEGA328P	[1.8V ; 5.5V]
LED Verte	> 2.5V
LED Infrarouge	> 1.6V
Potentiomètres	Synchronisée avec le MCU

Nous remarquons donc que les valeurs ne sont pas forcément compatibles avec la tension nominale de l'accumulateur. Rapidement nous remarquons aussi que la tension d'alimentation 5V est compatible avec tous les composants. Nous utiliserons un régulateur de tension 5V afin d'assurer une tension d'alimentation stable et compatible avec chacun des composants. Nous utiliserons un connecteur de 2 broches afin de relier l'accumulateur au circuit de la carte. Nous pourrons brancher le bloc énergie de la manière suivante :

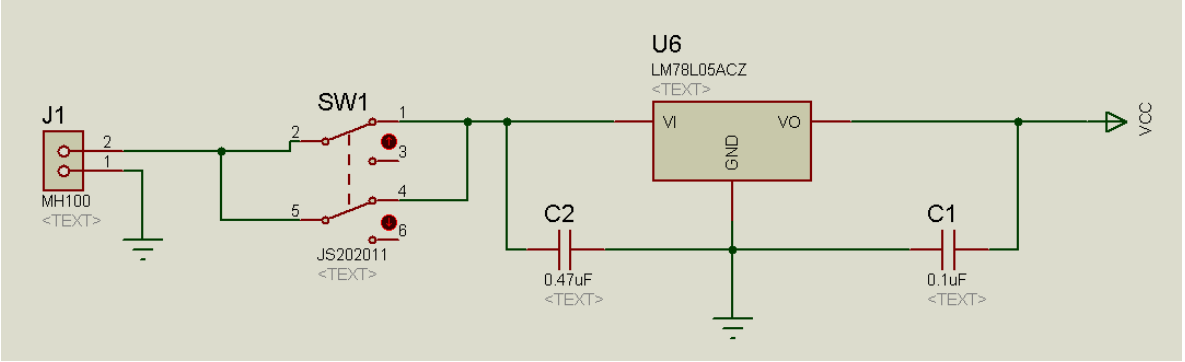


Figure 13 : Schéma électrique du régulateur linéaire branché en série avec l'accumulateur et l'interrupteur

Voir le choix de l'interrupteur dans le paragraphe suivant.

Référence du paragraphe : CPR_EMTT_INTERRUPTEUR

Rédacteur : Clément CACHO et Mathis BROUSSE

Relecteur : GRILLET Mathéo

Exigences client vérifiées par pré-conception : EXIG_INTERRUPTEUR

Compétences GEII : C1-10, C1-11

Le cahier des charges exige l'utilisation d'un interrupteur afin de mettre sous ou hors tension la carte de l'émetteur. Après lecture des datasheets nous avons constaté la présence de deux boutons poussoirs et un interrupteur. Nous utiliserons alors l'interrupteur JS202011CQN.

Référence du paragraphe : CPR_EMTT_SCHEMA

Rédacteur : Clément Cacho Mathéo Grillet Mathis Brousse

Relecteur : Clément Cacho Mathéo Grillet Mathis Brousse

Compétences GEII : C1-10, C1-11

Voir ci-dessous le schéma électrique préliminaire du produit complet.

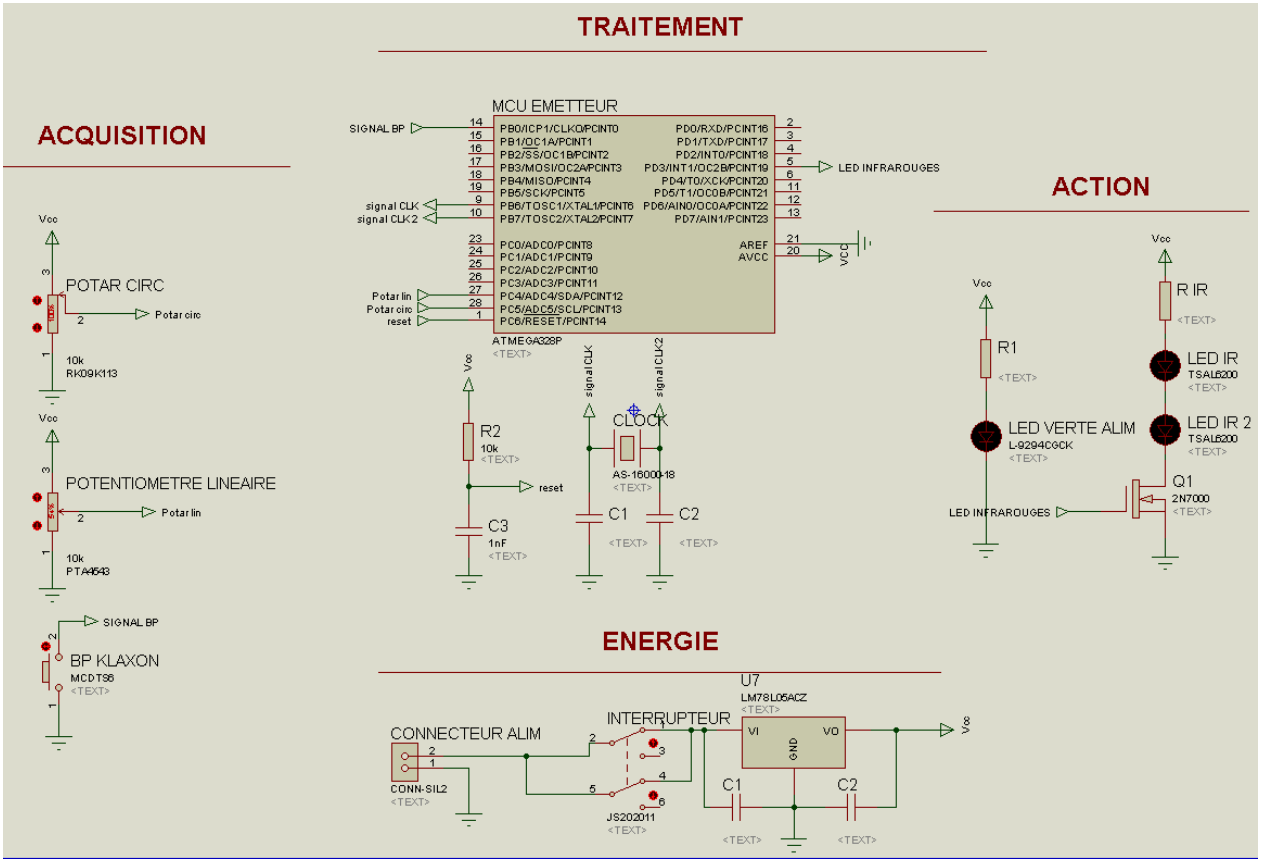


Figure 14 : Schéma électrique final de la conception préliminaire

2.2.2 Électronique - Récepteur

Référence du paragraphe : CPR_RCPT_ARCHI_ELEC

Rédacteur : DELANNOY Ylhan CORDEAU Maxence TISSOT Nicolas GIBELIN Thomas

Relecteur : DELANNOY Ylhan CORDEAU Maxence TISSOT Nicolas GIBELIN Thomas

Compétences GEII : C1-3, C1-9, C1-11

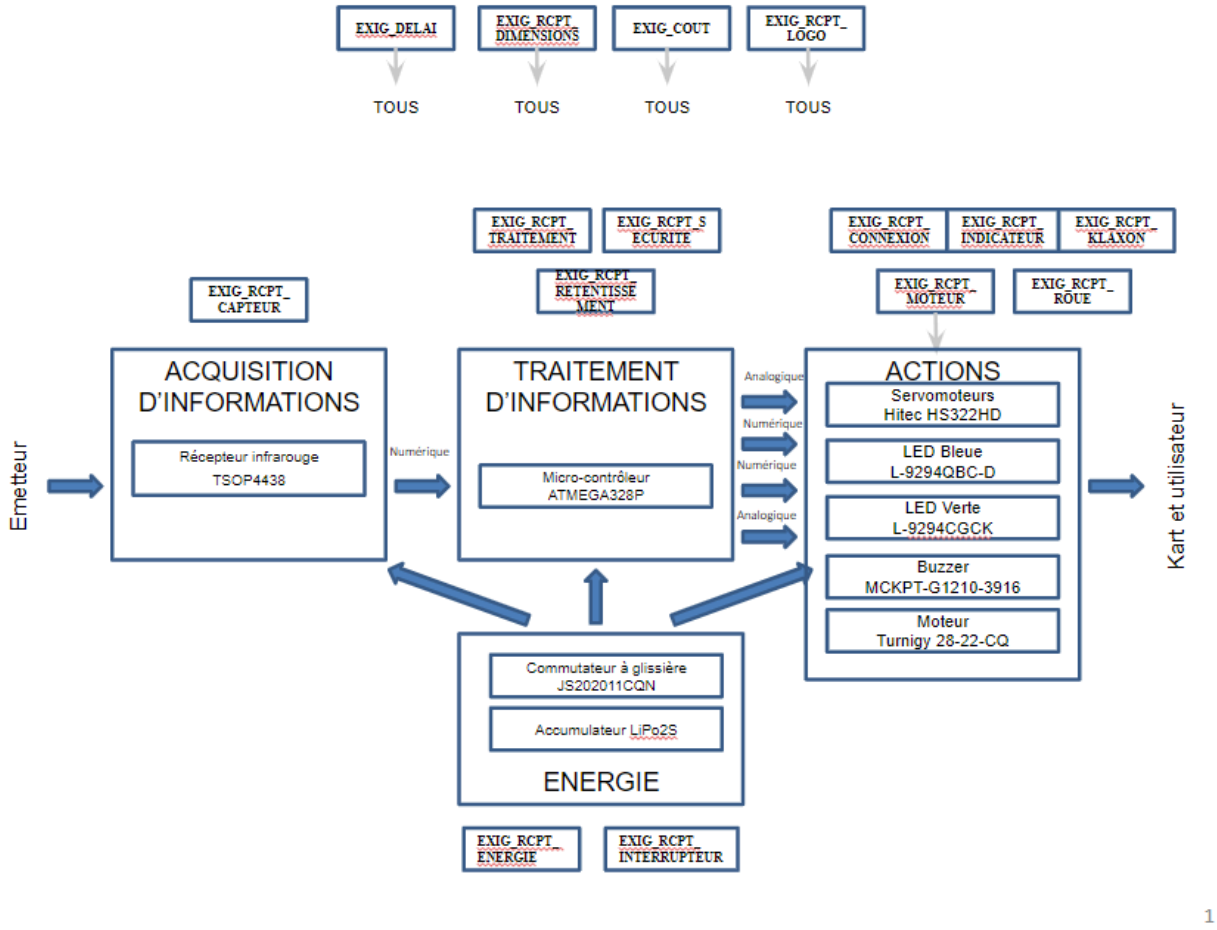


Figure 15 : synoptique architecture électronique du récepteur

Référence du paragraphe : CPR_RCPT_CAPTEUR

Rédacteur : DELANNOY Ylhan CORDEAU Maxence

Relecteur : TISSOT Nicolas GIBELIN Thomas

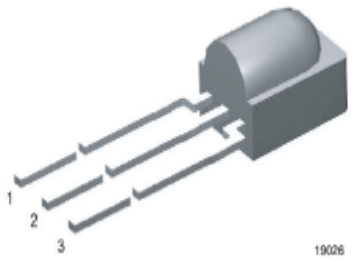
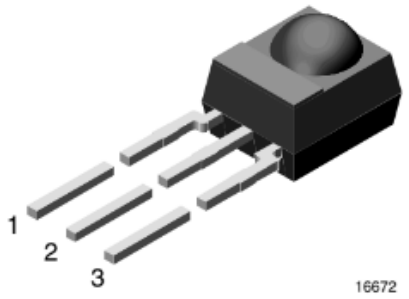
Exigences client vérifiées par pré-conception : EXIG_RCPT_CAPTEUR

Compétences GEII : C1-10, C1-11

Étage d'acquisition :

Dans cette section, notre objectif était de répondre à l'exigence visant à récupérer les données transmises par l'émetteur pour permettre au module de traitement de récupérer le signal NEC, nécessaire au déplacement du Kart sans rencontrer de problèmes.

En étudiant les spécifications techniques, nous avons identifié deux options pour le choix du capteur infrarouge dans la partie acquisition du récepteur. Initialement, en examinant les fiches techniques, nous avons constaté que les deux capteurs présentaient des caractéristiques générales similaires. Cependant, une différence significative réside dans le boîtier cylindrique du TSOP58438, qui peut ne pas être idéal pour notre application. Étant donné que le capteur est positionné en hauteur au-dessus du Kart, il est préférable d'opter pour un capteur de forme semi-sphérique, permettant une détection du signal dans toutes les directions. C'est pourquoi nous avons décidé d'adopter le récepteur TSOP4438, qui offre cette capacité. Cette option nous assure une détection efficace du signal, quel que soit l'angle de réception, ce qui est crucial pour le bon fonctionnement du système.

	
TSOP58438	TSOP4438

Nous avons donc récupéré quelques informations essentielles à propos de ce capteur infrarouge.

Tension de fonctionnement	2.5V à 5.5V
Fréquence de fonctionnement	38kHz
3 broches	1 = OUT, 2 = GND, 3 = VS

En conséquence nous avons donc un récepteur infrarouge ayant pour angle de fonctionnement de 180° en forme de demi sphère, qui sera situé en dehors de la carte électronique sur le kart.

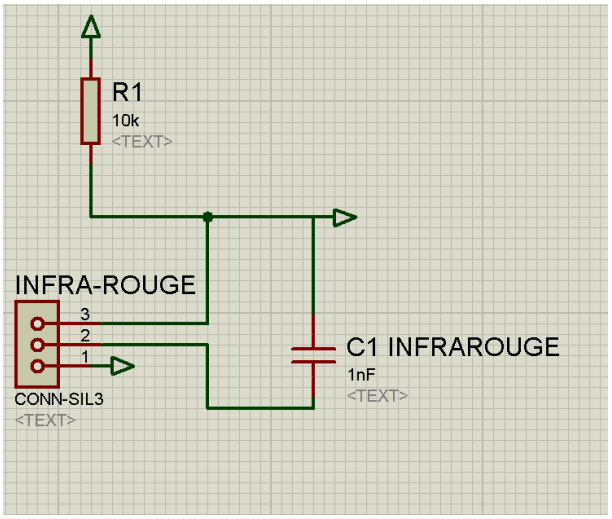


Figure 16: Schéma électrique préliminaire du récepteur infrarouge

Référence du paragraphe : CPR_RCPT_TRAITEMENT

Rédacteur : TISSOT Nicolas, GIBELIN Thomas

Relecteur : DELANNOY Ylhan CORDEAU Maxence

Exigences client vérifiées par pré-conception : EXIG_RCPT_TRAITEMENT

Compétences GEII : C1-10, C1-11

Afin de respecter l'exigence demandée dans la cadre du bloc de traitement de récepteur. Nous avons besoin d'un microcontrôleur qui permettra de gérer les informations reçues et à transmettre.

Pour le choix du microcontrôleur, nous avons besoin de 8 bits pour le déplacement du Kart, 4 bits pour la puissance du moteur et 4 bits pour la direction des roues.

Plusieurs contrôleurs sont à notre disposition, mais seul l'ATMEGA 328P correspond à nos besoins.

En effet celui-ci est codé sur 8 bits contrairement aux autres qui sont codés sur 32 bits. De plus, il possède 28 broches qui permettent de réduire la place nécessaire sur le circuit imprimé. Et enfin l'ATMEGA 328P est le moins chère avec 2.62 € HT. Toutes ces caractéristiques font de lui la meilleure option dans le choix des microcontrôleurs.

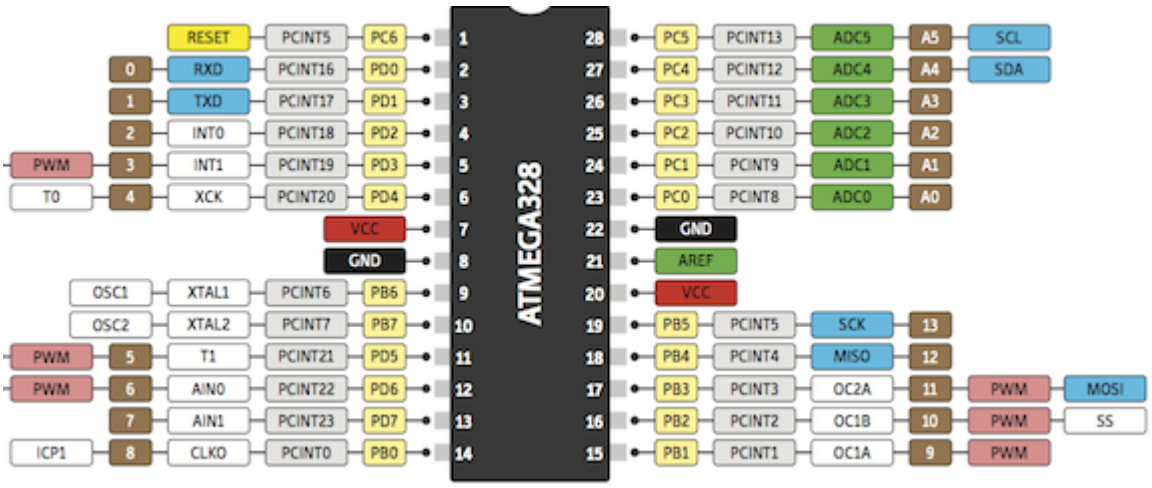


Figure 17 : Schéma électrique de l'ATMEGA 328P

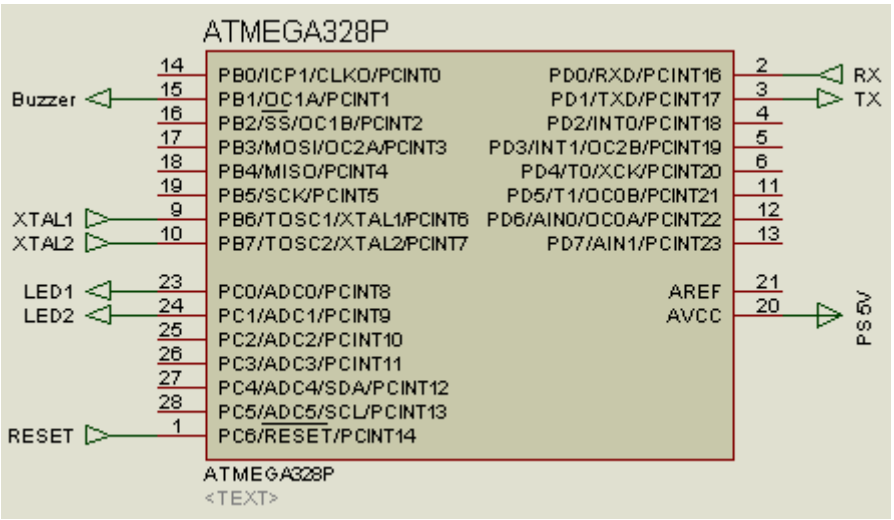


Figure 18 : Schéma électrique de l'ATMEGA 328P sur ISIS

Référence du paragraphe : CPR_RCPT_SECURITE

Rédacteur : TISSOT Nicolas, GIBELIN Thomas

Relecteur : Delannoy ylhan Cordeau Maxence

Exigences client vérifiées par pré-conception : EXIG_RCPT_SECURITE

Compétences GEII : C1-10, C1-11

Pour permettre la mesure d'une absence de signal infrarouge ou d'une réception invalide de signal infrarouge, nous avons choisi d'utiliser un timer qui est compris dans le micro-contrôleur.

Référence du paragraphe : CPR_RCPT_RETENTISSEMENT

Rédacteur : TISSOT Nicolas, GIBELIN Thomas

Relecteur : Delannoy ylhan Cordeau Maxence

Exigences client vérifiées par pré-conception : EXIG_RCPT_RETENTISSEMENT

Compétences GEII : C1-10, C1-11

En fonction de la trame NEC reçue, le micro-contrôleur déterminera s'il faut ou non déclencher le buzzer.

Référence du paragraphe : CPR_RCPT_MOTEUR

Rédacteur : DELANNOY Ylhan CORDEAU Maxence

Relecteur : TISSOT Nicolas, GIBELIN Thomas

Exigences client vérifiées par pré-conception : EXIG_RCPT_MOTEUR

Compétences GEII : C1-10, C1-11

Pour contrôler les hélices de notre kart, un moteur est nécessaire. Le moteur doit être autonome sur sa rotation. Nous relevons un moteur brushless, le Turnigy 28-22-CQ 1400Kv qui est connecté à l'aide de 3 broches (l'alimentation "+", l'alimentation "-") ainsi que la broche envoyant le signal PWM qui contrôle la vitesse en fonction du rapport cyclique du signal. Le moteur brushless est précédé d'un contrôleur BEC qui premièrement permet d'augmenter le courant de sortie et dans un second temps sert aussi à réguler la tension d'entrée pour avoir un moteur stable. La partie traitement est directement impliquée dans le fonctionnement du moteur puisqu'en fonction du signal délivré la vitesse du moteur varie.

Caractéristiques du Turnigy 28-22-CQ 1400Kv :

Courant d'entrée minimum	5A
Courant d'entrée maximum	8A
Tension d'entrée	6~9V

Référence du paragraphe : CPR_RCPT_ROUE

Rédacteur : DELANNOY Ylhan CORDEAU Maxence

Relecteur : TISSOT Nicolas, GIBELIN Thomas

Exigences client vérifiées par pré-conception : EXIG_RCPT_ROUE

Compétences GEII : C1-10, C1-11

Nous désirons pouvoir contrôler la direction du Kart à hélices en utilisant un servomoteur.

Ce dernier est chargé de réguler l'angle de direction à l'avant du kart à hélice. Pour ce faire, il est nécessaire de moduler le rapport cyclique du servomoteur : un rapport cyclique faible correspond à un angle de 0°, tandis qu'un rapport cyclique élevé correspond à un angle de 180°, permettant ainsi de diriger le kart vers la gauche (0°) ou vers la droite (180°).

Par ailleurs, le servomoteur est connecté à l'unité de traitement par trois câbles : le premier pour l'alimentation positive (+), le deuxième pour la mise à la terre (-) et le troisième pour le signal PWM, qui permet de contrôler le servomoteur en lui envoyant les instructions appropriées.

En conséquence, nous avons opté pour le servomoteur HS-322HD.

Informations à propos du composants du HS-322HD:

Tension de fonctionnement	4.8 à 6.0V
Courant de fonctionnement maximum	7.4mA

Référence du paragraphe : CPR_RCPT_INDICATEUR

Rédacteur : DELANNOY Ylhan CORDEAU Maxence

Relecteur : TISSOT Nicolas, GIBELIN Thomas

Exigences client vérifiées par pré-conception : EXIG_RCPT_INDICATEUR

Compétences GEII : C1-10, C1-11

Notre carte électronique a pour rôle, d'après le cahier des charges, d'indiquer à l'utilisateur lorsque la carte est sous tension à l'aide d'une LED qui est le plus souvent de couleur verte.

Nous repérons le composant L-9294CGCK qui remplit parfaitement ce rôle.

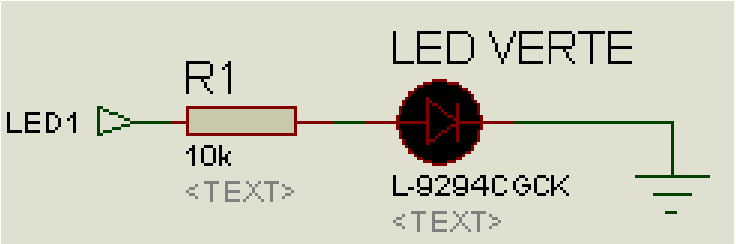


Figure 19 : Schéma électrique préliminaire de la LED verte

Référence du paragraphe : CPR_RCPT_CONNEXION

Rédacteur : DELANNOY Ylhan CORDEAU Maxence

Relecteur : TISSOT Nicolas, GIBELIN Thomas

Exigences client vérifiées par pré-conception : EXIG_RCPT_CONNEXION

Compétences GEII : C1-10, C1-11

D'après le cahier des charges, la carte électronique du récepteur doit être équipée d'un indicateur lumineux indiquant la réception d'une trame NEC depuis l'émetteur. Par convention nous choisissons la couleur bleue. Nous utiliserons donc une LED de couleur bleue, la L-9294QBC-D qui satisfait pleinement l'exigeance.

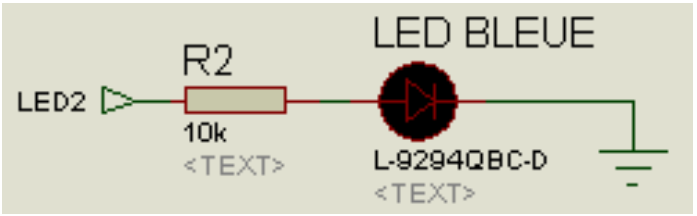


Figure 20 : Schéma électrique préliminaire de la LED bleue

Référence du paragraphe : CPR_RCPT_KLAXON

Rédacteur : DELANNOY Ylhan CORDEAU Maxence

Relecteur : TISSOT Nicolas, GIBELIN Thomas

Exigences client vérifiées par pré-conception : EXIG_RCPT_KLAXON

Compétences GEII : C1-10, C1-11

La carte électronique est équipée d'un buzzer qui émet le son du klaxon, nous choisissons le composant MCKPT-G1210-3916. Il est capable d'émettre un son à une fréquence de 4,5kHz à +/- 0.5kHz soit plus de 4kHz comme spécifié dans le cahier des charges. Il fonctionne avec un signal carré en entrée compris en 3 et 30Vpp pour un courant max de 2mA.

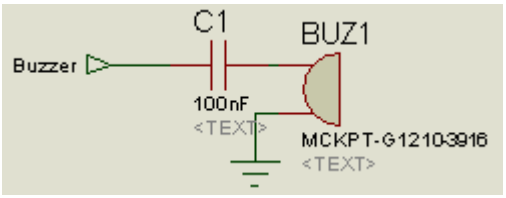


Figure 21: Schéma électrique préliminaire du buzzer

Référence du paragraphe : CPR_RCPT_ENERGIE

Rédacteur : TISSOT Nicolas, GIBELIN Thomas

Relecteur : DELANNOY Ylhan CORDEAU Maxence

Exigences client vérifiées par pré-conception : EXIG_RCPT_ENERGIE

Compétences GEII : C1-10, C1-11

Nous utilisons un accumulateur Lipo 2S qui fournit une tension nominale de 7.4V. Or, certains composants ne supportent pas une tension supérieure à 5V. Pour permettre de contrôler le moteur brushless, nous utilisons un contrôleur BEC qui fournit une tension de 5V à celui-ci mais également au reste des composants, ce qui nous dispense l'utilisation d'un régulateur de tension.

Composants	Tension de fonctionnement maximum
Capteur Infrarouge	2.5V-5.5V
ATMEGA328P	5.5V
Servomoteur Hitec HS322HD	4.8V-6.0V

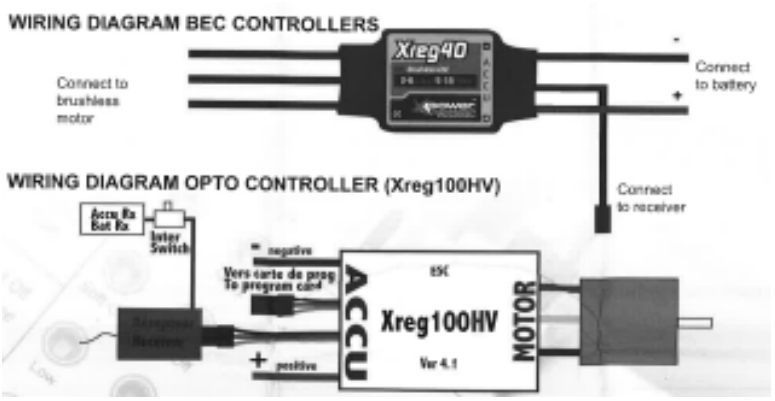


Figure 22 : Schéma électrique préliminaire du contrôleur BEC

Référence du paragraphe : CPR_RCPT_INTERRUPTEUR

Rédacteur : TISSOT Nicolas, GIBELIN Thomas

Relecteur : Delannoy ylhan Cordeau Maxence

Exigences client vérifiées par pré-conception : EXIG_RCPT_INTERRUPTEUR

Compétences GEII : C1-10, C1-11

D'après le cahier des charges, la carte électronique du récepteur doit être équipée d'un interrupteur qui permet de mettre sous/hors tension la carte. Nous faisons le choix d'un bouton poussoir R1825A car celui-ci possède une meilleure ergonomie avec une taille plus importante que les autres choix disponibles. Il sera donc plus simple pour l'utilisateur d'appuyer sur celui-ci. On fait le choix de la placer à l'arrière du Kart pour être facilement accessible.

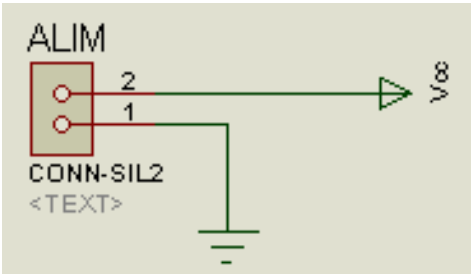


Figure 23: schéma électrique préliminaire du bouton poussoir

Figure 23 :*Bouton poussoir R1825A*

Référence du paragraphe : CPR_RCPT_SCHEMA

Rédacteur : TISSOT Nicolas, GIBELIN Thomas DELANNOY Ylhan CORDEAU Maxence

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Compétences GEII : C1-10, C1-11

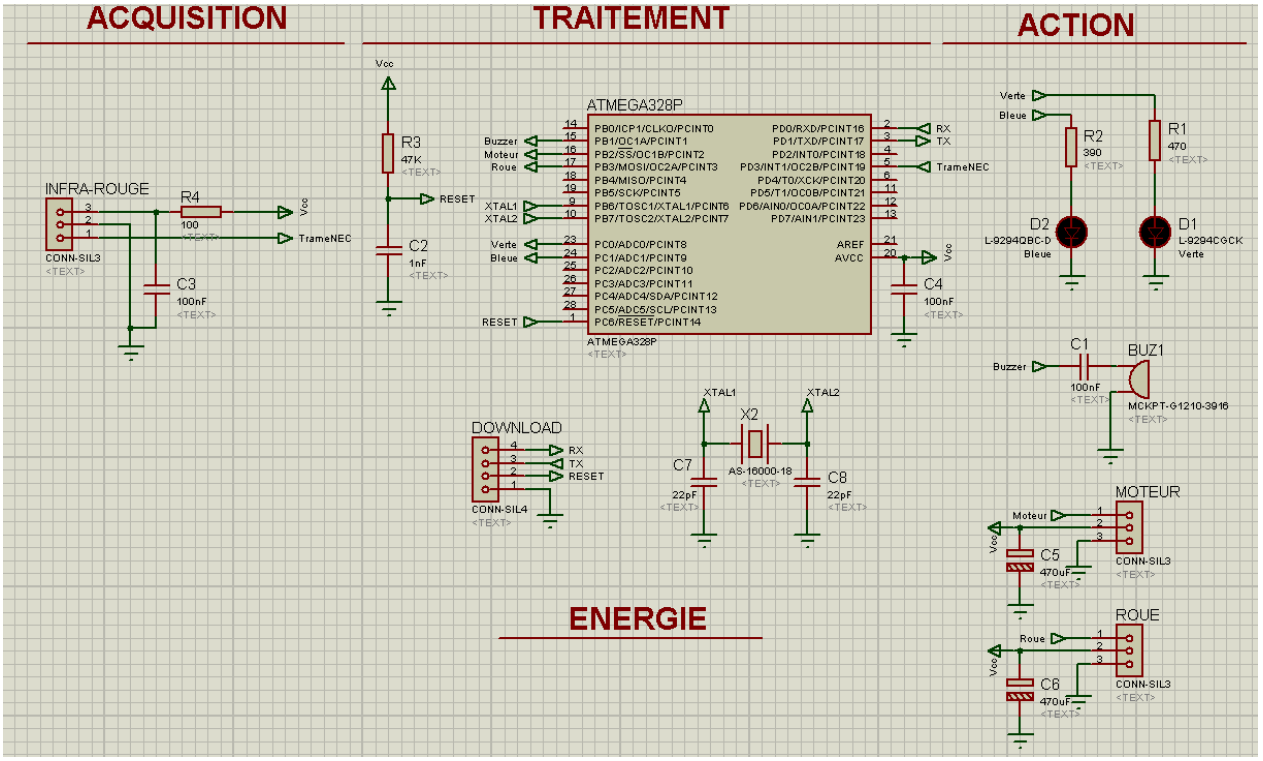


Figure 24 : schéma électrique préliminaire du récepteur

2.3 Informatique

2.3.1 Informatique - Émetteur

Référence du paragraphe : CPR_EMTT_ARCHI_INFO

Rédacteur : TISSOT Nicolas, GIBELIN Thomas DELANNOY Ylhan CORDEAU Maxence

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Compétences GEII : C1-3, C1-9, C1-10, C1-11

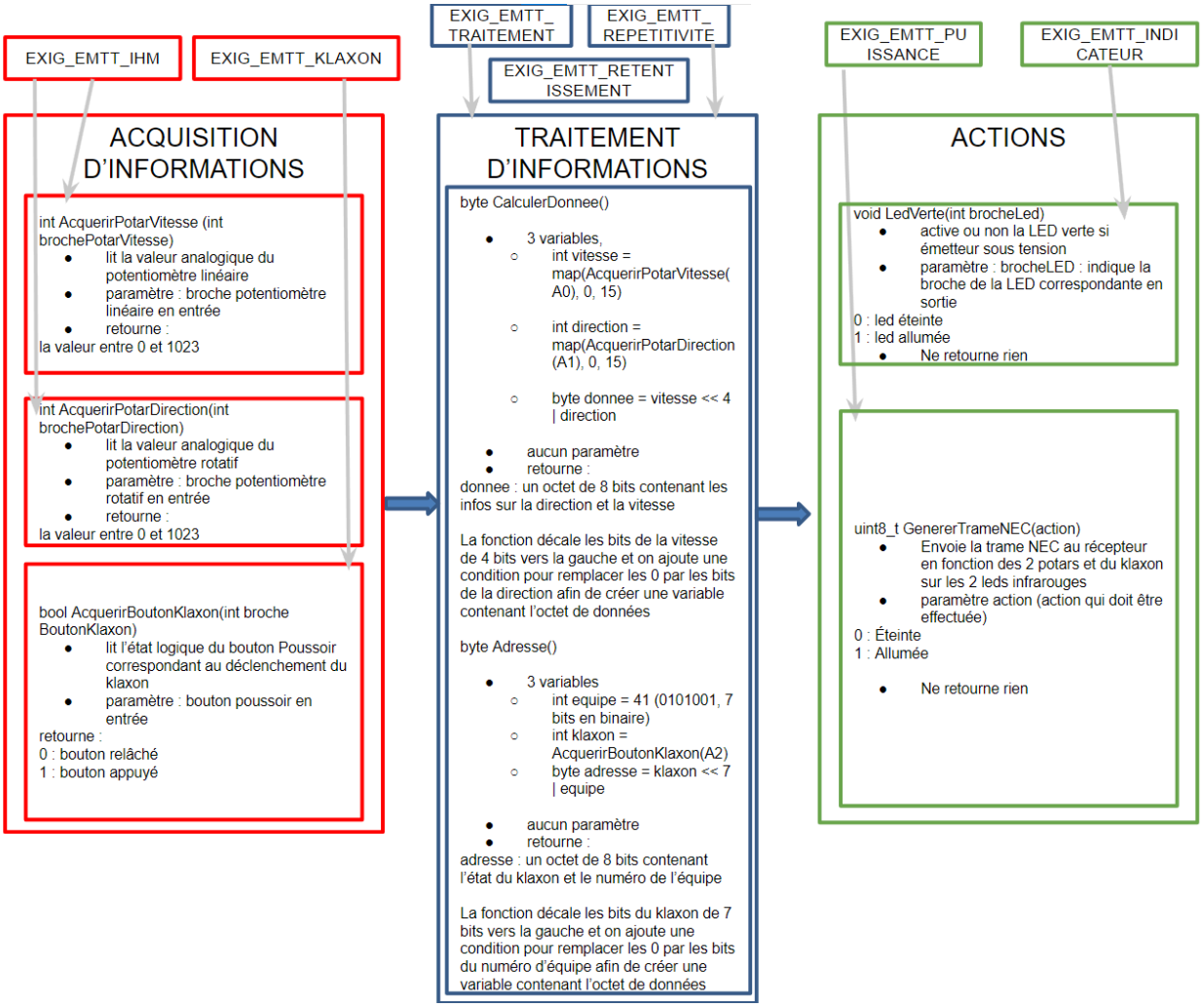


Figure 25 : synoptique architecture informatique de l'émetteur

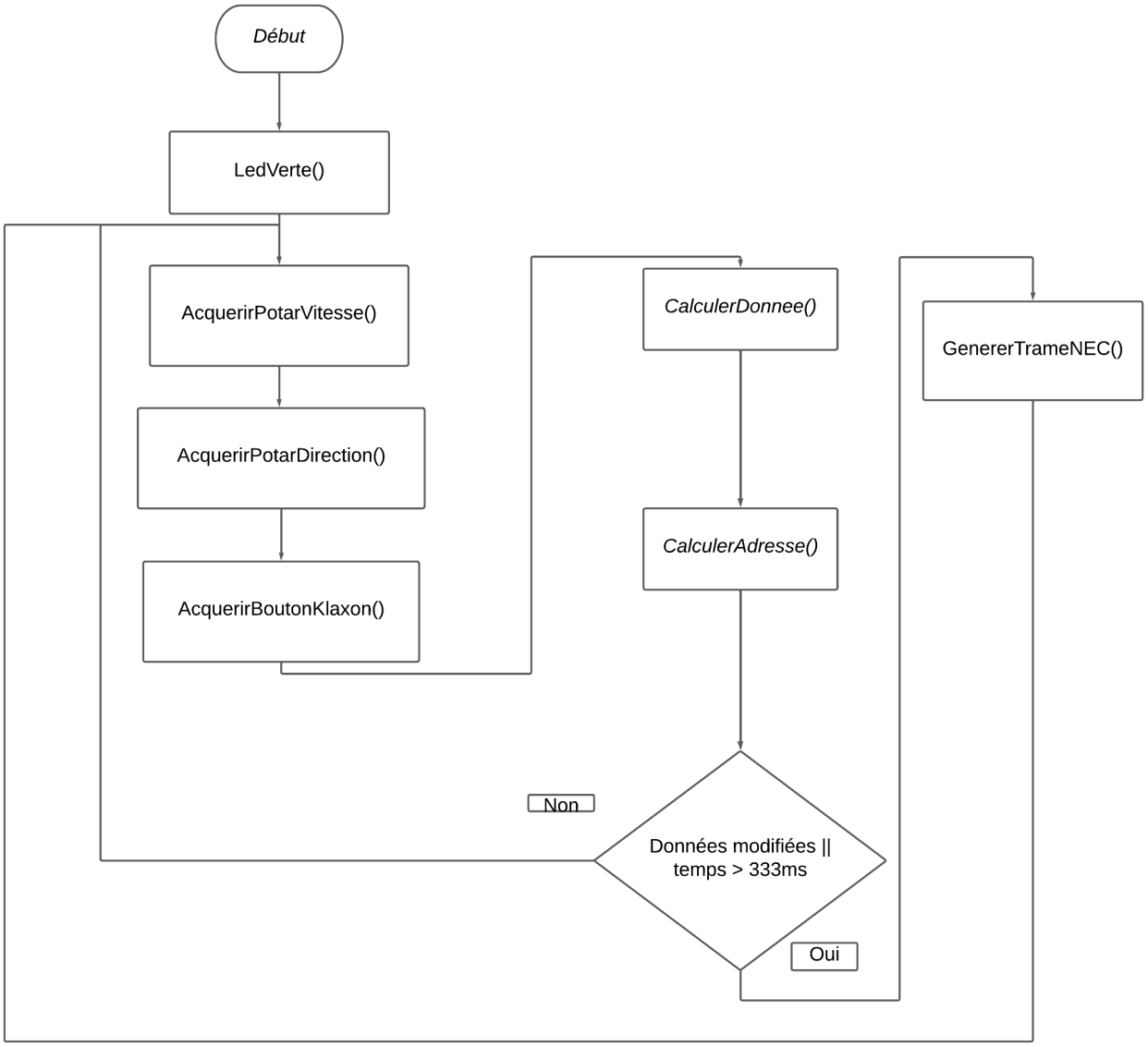


Figure 26 : algorigramme de traitement de l'émetteur

2.3.2 Informatique - Récepteur

Référence du paragraphe : CPR_RCPT_ARCHI_INFO

Rédacteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo

Relecteur : TISSOT Nicolas, GIBELIN Thomas DELANNOY Ylhan CORDEAU Maxence

Compétences GEII : C1-3, C1-9, C1-10, C1-11

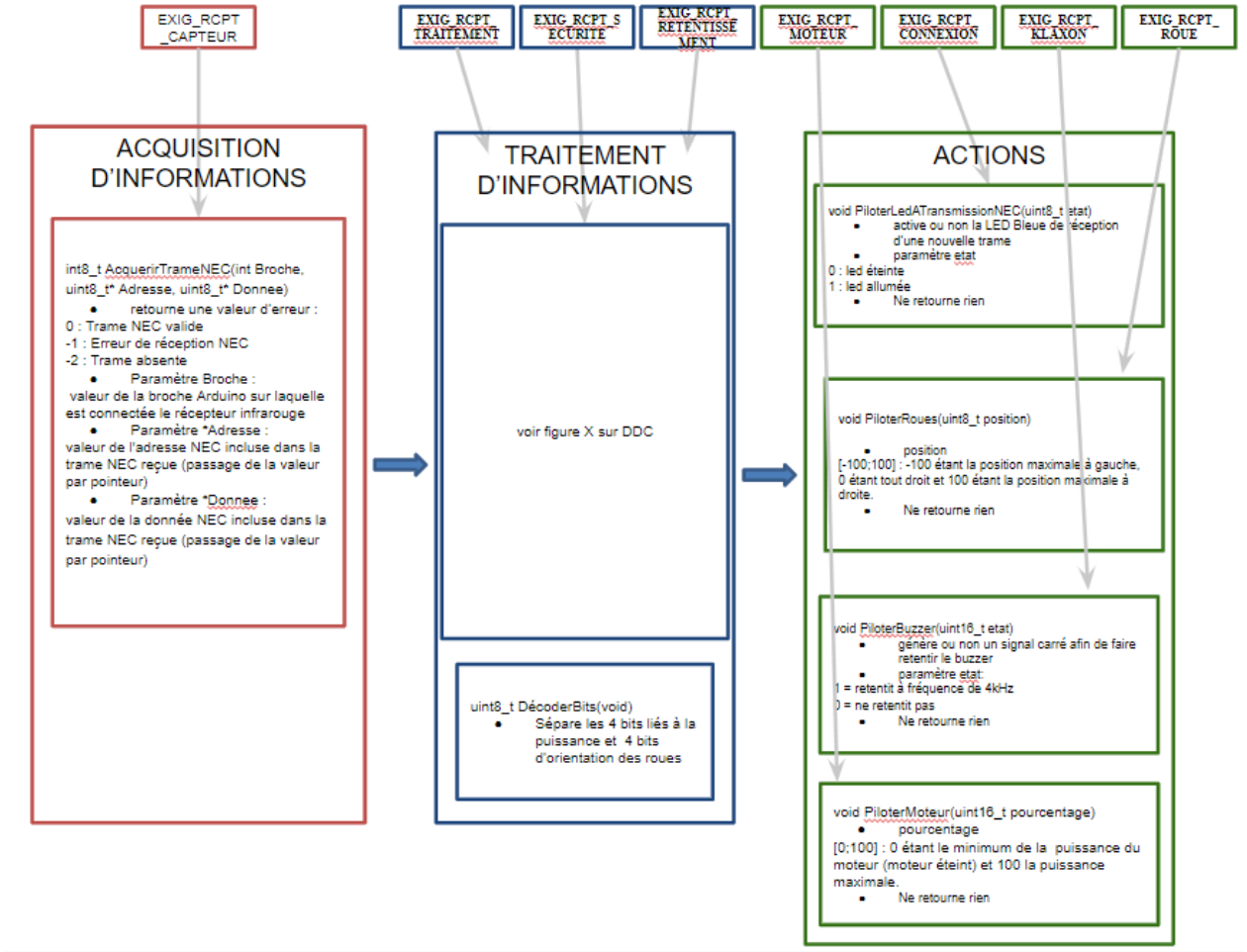


Figure 27 : synoptique architecture informatique du récepteur

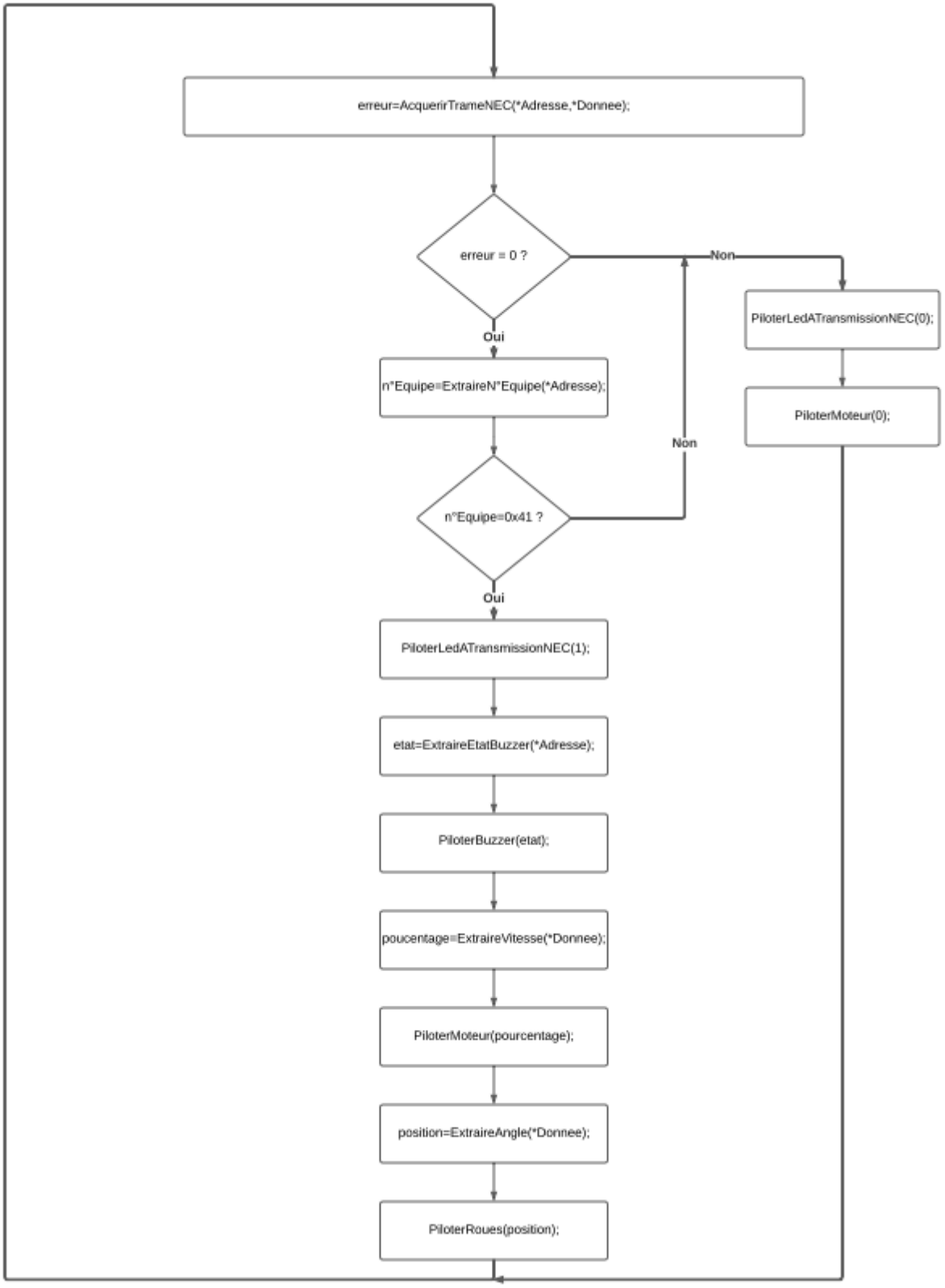


Figure 28 : organigramme de traitement du récepteur

2.4 Coût - Délai

Référence du paragraphe : CPR_COUT

Rédacteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo TISSOT Nicolas, GIBELIN Thomas, Maxence CORDEAU, Ylhan DELONNOY

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo TISSOT Nicolas, GIBELIN Thomas, Maxence CORDEAU, Ylhan DELONNOY

Exigences client vérifiées par pré-conception : EXIG_COUT

Compétences GEII : C1-10

Voici le coût de notre projet après la conception préliminaire :

Coût du projet

Référence du paragraphe : CPR_DELAI

Rédacteur : Mathis Brousse et Clément Cacho

Relecteur : Mathéo GRILLET

Exigences client vérifiées par pré-conception : EXIG_DELAI

Compétences GEII : C1-10

Nous avons suivi et tenu le planning initial.

Voici le lien de ce planning :

Planning

2.5 Conclusion de la conception préliminaire du produit

Rédacteur : TISSOT Nicolas, GIBELIN Thomas

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo, Maxence CORDEAU, Ylhan DELONNOY

Suite à la fin de la conception préliminaire, nous pouvons conclure sur la faisabilité du projet Kart à Hélice. Nous pouvons affirmer que le coût est inférieur à 160 € TTC et que la remise du dossier de conception préliminaire est conforme au délai imposé par le planning.

Nous pouvons donc commencer la conception détaillée.

Conception détaillée du produit

Ce chapitre détaille la conception du produit développé. Il constitue une preuve de la conformité du produit. Chaque paragraphe de cette étude fait donc clairement référence aux exigences client issues du [CDC].

3.1 Mécanique

3.1.1 Mécanique - Émetteur

Pour l'émetteur, certains composants ont changé de place par rapport au plan mécanique établi en conception préliminaire. Nous avons notamment décidé de changer le klaxon de côté afin de nous faciliter le routage. Cela ne change en rien voire augmente l'ergonomie de notre carte car la plupart des personnes sont droitières. Le klaxon est donc plus facile d'accès à l'aide de notre pouce. Les deux potentiomètres et les deux LED infrarouges sont restés à la même place par rapport au plan mécanique établi en conception préliminaire. Nous avons alors le routage suivant :

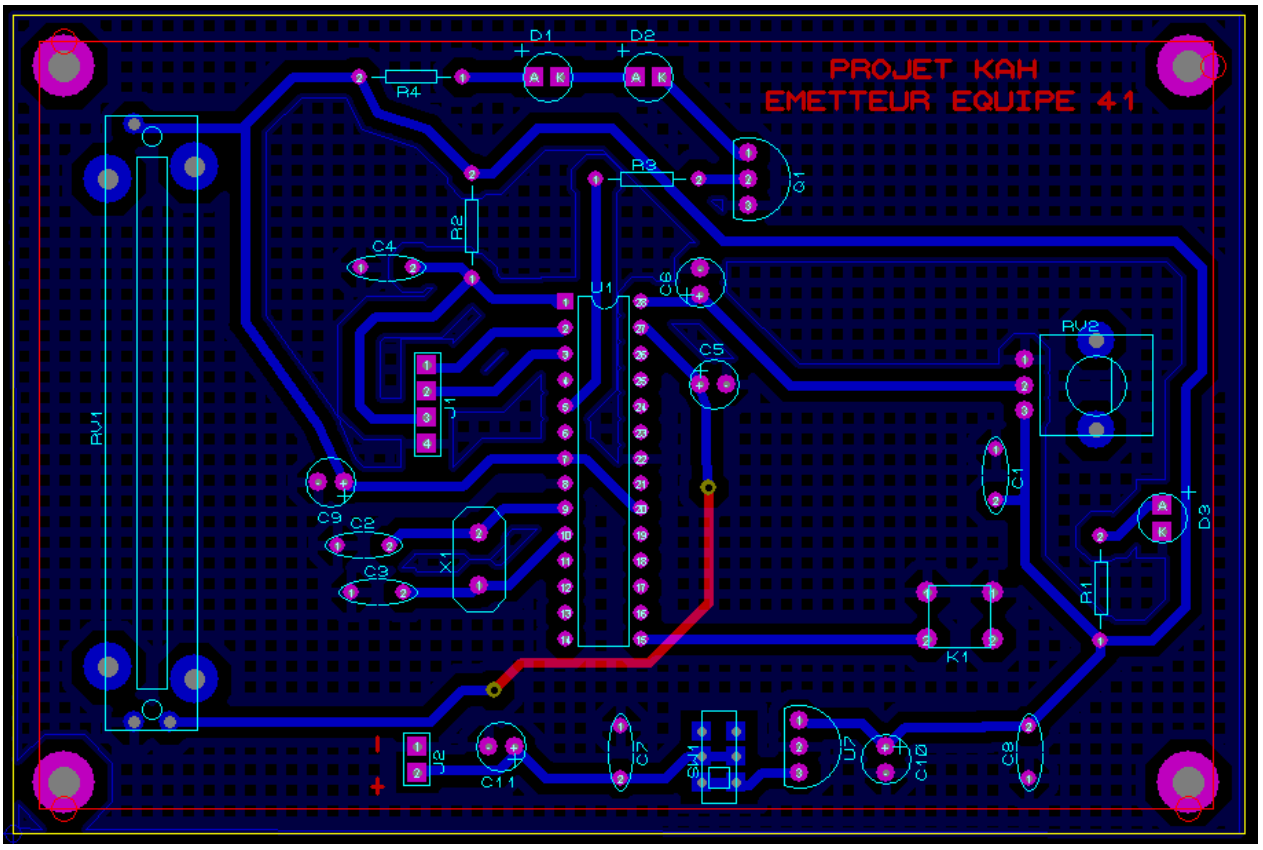


Figure 29 :Routage de la carte émetteur

3.1.2 Mécanique - Récepteur

Suite au routage ainsi qu'à la conception détaillée, nous avons ajouté des composants et modifié l'emplacement de certains déjà existant afin d'améliorer l'ergonomie de la carte.

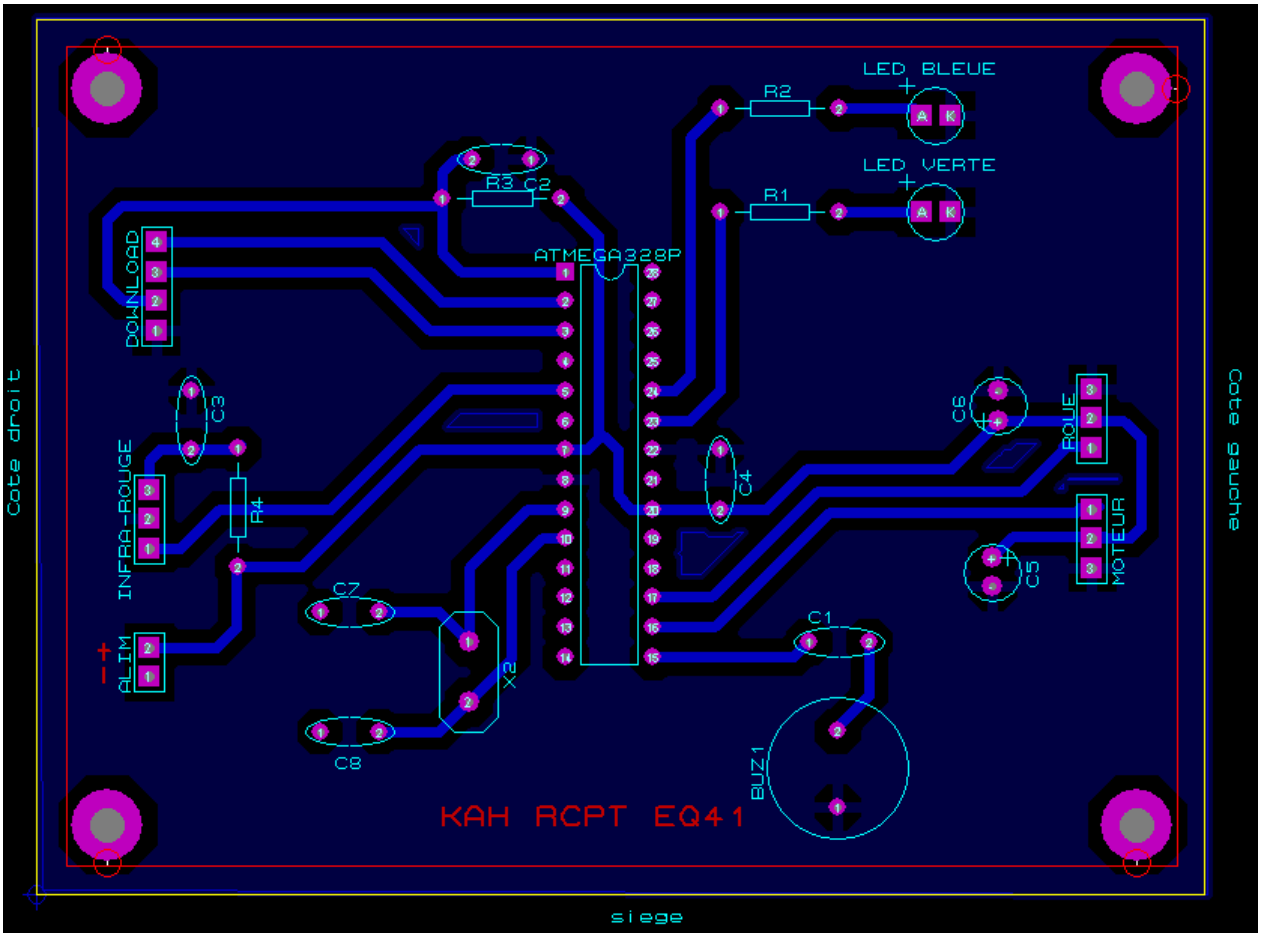


Figure 30 : routage de la carte récepteur

Nous avons décidé d'ajouter un connecteur 4 pins afin de permettre le téléchargement du programme placé à droite de la carte. Nous avons également ajouté 3 condensateurs de découplage placé proches des composants concernés.

Afin d'améliorer l'ergonomie de la carte ainsi que la facilité de routage, nous avons appliqué au microprocesseur une rotation. Les connecteurs des roues et moteur ont été descendu tout en restant à gauche afin de permettre aux câbles d'alimentation et de connections de se brancher sans difficultés.

3.2 Electronique

3.2.1 Emetteur

Référence du paragraphe : CDT_EMTT_IHM

Rédacteur : Mathéo GRILLET

Relecteur : Mathis BROUSSE et Clément CACHO

Exigences client vérifiées : EXIG_EMTT_IHM

Compétences GEII : C1-21,22,23,24,25,26

Nous allons déterminer la valeur du potentiomètre de sorte à obtenir la plus grande précision possible avec le MCU. C'est-à-dire une valeur codée sur 4 bits. Grâce à la formule (1) nous pouvons en déduire le nombre de possibilités.

$$n = 2^{nbits} \quad (1)$$

Par application numérique nous trouvons donc 16 valeurs possibles. Nous souhaitons donc une précision supérieure à 1/16 soit 6.26%. La précision est fonction du rapport entre le courant i (notation de la figure 29) trouvé par lecture de la datasheet et le courant Ipont.

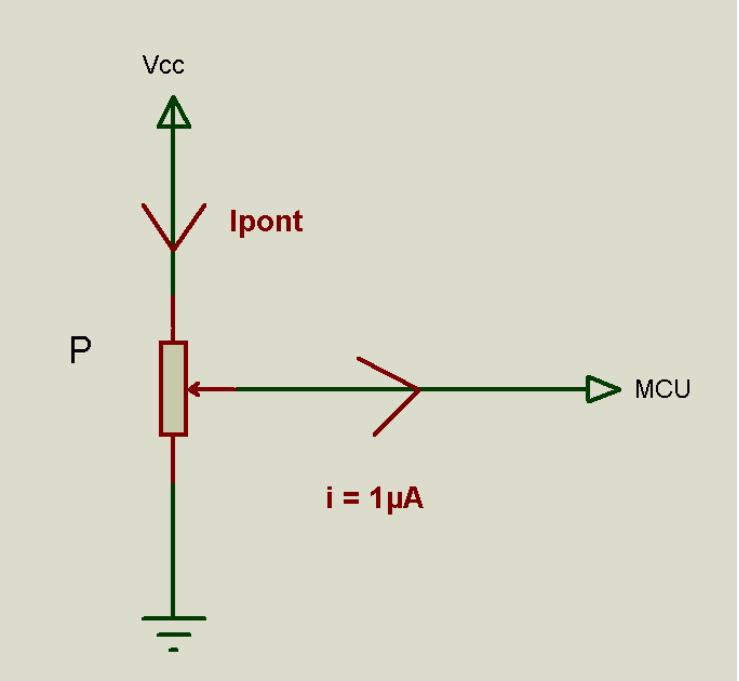


Figure 31 : schéma électrique du potentiomètre annoté

Nous en déduisons la courant Ipont minimal:

$$précision = i \div Ipont$$

$$Ipont > 16\ \mu A$$

Nous pouvons calculer la valeur de la résistance du potentiomètre :

$$R < \frac{Vcc}{Ipont}$$

$$R < 312\,500$$

Les deux potentiomètres (rotatifs et rectilignes) de 10kΩ auront donc une précision supérieure à 6.25%.

Afin de réduire au maximum tout problème nous allons ajouter un condensateur comme représenté sur la figure 30.

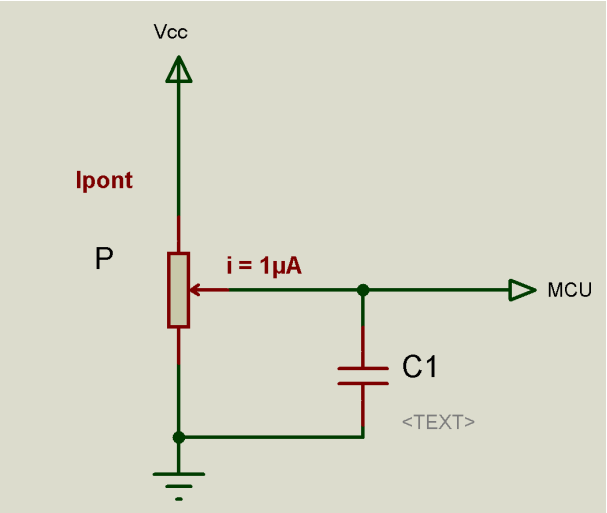


Figure 32 : schéma électrique du potentiomètre avec le condensateur

On estime que les variations de tensions sont intéressantes lorsqu'elles sont supérieures à 5Hz (fréquence maximale à laquelle nous allons déplacer le potentiomètre). Au delà de 5Hz il s'agit de bruit.

Selon la formule de Shannon:

$$Fc = \frac{1}{2\pi.P.C}$$

Avec Fc = 5Hz

P= 10 000 Ω

Nous pouvons donc en déduire C:

$C = \frac{1}{2\pi.P.Fc} = 3.18\ \mu F$

On normalise la valeur du composant à 4,7µF.

Référence du paragraphe : CDT_EMTT_KLAXON

Rédacteur : Mathéo GRILLET

Relecteur : Mathis BROUSSE et CACHO Clément

Exigences client vérifiées : EXIG_EMTT_KLAXON

Compétences GEII : C1-21,22,23,24,25,26

Afin de simplifier le routage nous branchons le bouton en pull-up (voir figure 31). Nous utiliserons le paramètre “INPUT_PULLUP” lors de la programmation.

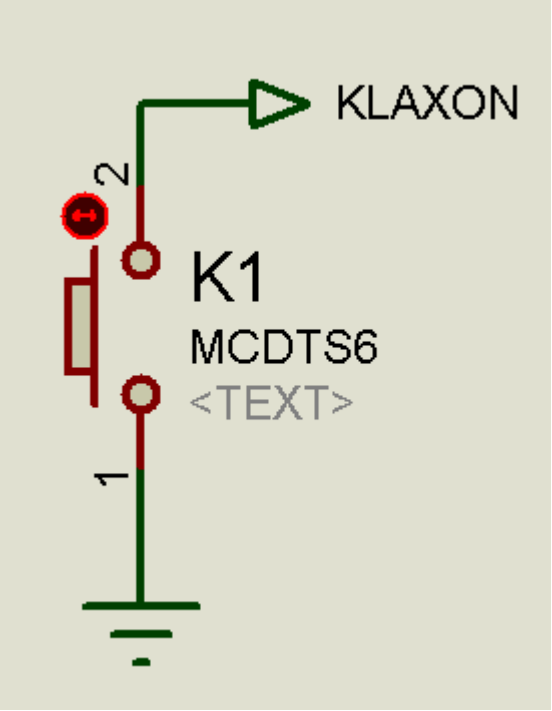


Figure 33: Schéma de câblage du bouton

L’exigence nous demandant de placer le bouton du klaxon de manière ergonomique, nous choisissons de placer celui-ci proche du potentiomètre rotatif sur la partie droite de la carte.

Référence du paragraphe : CDT_EMTT_TRAITEMENT

Rédacteur : Mathis BROUSSE et CACHO Clément

Relecteur : Mathéo GRILLET

Exigences client vérifiées :EXIG_EMTT_TRAITEMENT

Compétences GEII : C1-21,22,23,24,25,26

Nous allons détailler dans cette partie la conception détaillée des composants permettant le bon fonctionnement du microcontrôleur afin de gérer l'exigence liée au traitement. Pour ce faire nous détaillerons les composants liés au reset du microcontrôleur. (voir figure 32)

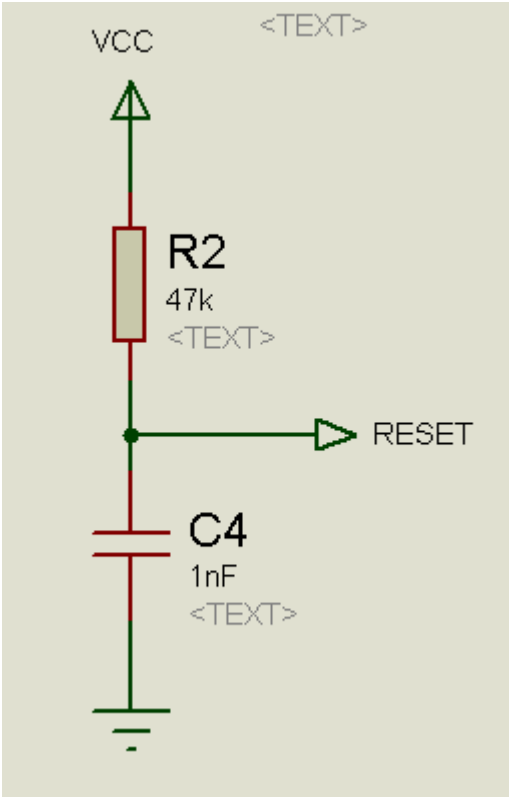


Figure 34 : Schéma électrique de l'étage reset du MCU

Nous devons donc dimensionner le condensateur et la résistance permettant de générer le signal reset. Nous avons extrait de la datasheet la valeur du condensateur : C3 = 1nF. De la même manière nous avons extrait la valeur de $V_{IL} = 0.1V_{cc}$ ainsi que $t_{reset} > 2.5\mu s$. Pour sélectionner ces valeurs nous nous sommes placés dans le cas où nous avons le pire des composants. Or nous avons la relation de la tension aux bornes du condensateur :

$V_c(t) = V_{c\infty} - (V_{c\infty} - V_{cini})e^{-t/\tau}$ avec $V_{c\infty} = V_{cc}$; $V_{cini} = 0$ et $\tau = RC$.

Or nous savons que pour un temps t_{reset} la tension $V_c(t_{reset}) = V_{IL} = 0.1V_{cc}$

Nous pouvons donc réécrire notre relation : $V_c(t) = V_{cc}(1 - e^{-t/\tau})$

Donc nous pouvons dimensionner la résistance R par l'équation suivante :

$V_c\,(t_{reset}) = V_{IL} > V_{cc}\,(1 - e^{-treset/\tau})$

Après résolution de l'équation nous avons déterminé R > 24k. Nous allons donc dimensionner cette résistance par les méthodes générales de dimensionnement d'une résistance. La résistance devant être supérieure à une valeur, nous n'avons pas de tolérance à respecter, nous pouvons donc utiliser une résistance de la série E3 pour des raisons économiques. Nous utiliserons donc une résistance R = 47kΩ. Afin de vérifier la compatibilité de cette valeur avec notre utilisation nous pouvons réaliser le calcul inverse, vérifier le temps de reset avec une telle valeur de résistance, nous repartons donc par :

$V_c\,(t_{reset}) = V_{IL} = V_{cc}\,(1 - e^{-treset/\tau})$ avec $\tau = RC$. Lorsque nous isolons t_{reset} on obtient $t_{reset} = 4.95\mu s$

Pour le téléversement et le débogage du programme, nous allons utiliser un dongle. Ce dernier est une carte électronique externe connectée en USB, qui permet de convertir une liaison USB en une liaison USART. Nous pouvons brancher ce composant avec un connecteur 4 broches afin de de téléverser et déboguer le programme. Cela nous permet de gagner de la place ainsi que de l'argent. C'est d'ailleurs ce système qui est utilisé pour le fonctionnement du jeu SIMON. Pour trouver cette alternative nous nous sommes inspirés de ce jeu ainsi qu'aidé de la ressources numéro 22 de GEII. (voir figure 35 et 36)

Figure 35 : Dongle de téléversement du programme vers le MCU

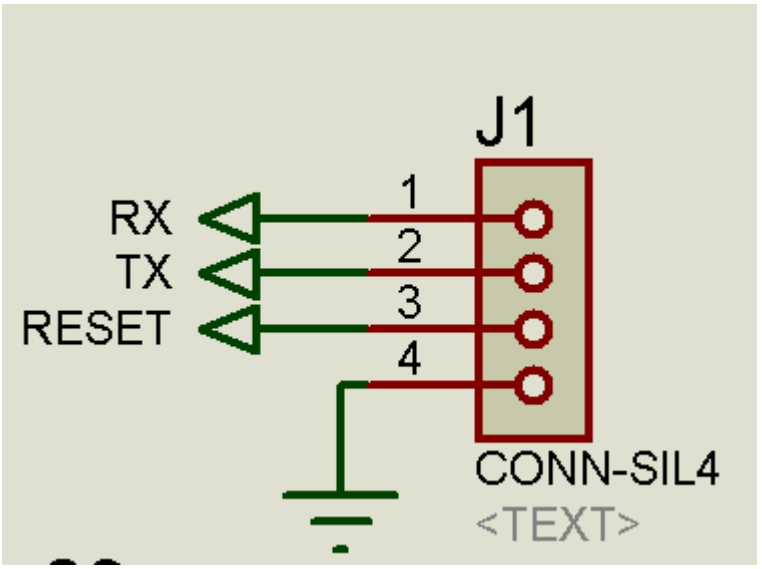


Figure 36 : Montage du connecteur 4 broches

De plus, les condensateurs que nous allons utiliser afin d'assurer le bon fonctionnement de l'horloge sont deux condensateurs de 22pF. Pour trouver ces valeurs, nous avons pris exemple sur le jeu simon ainsi que sur la ressource du numéro 22 du GEII. Tel que :

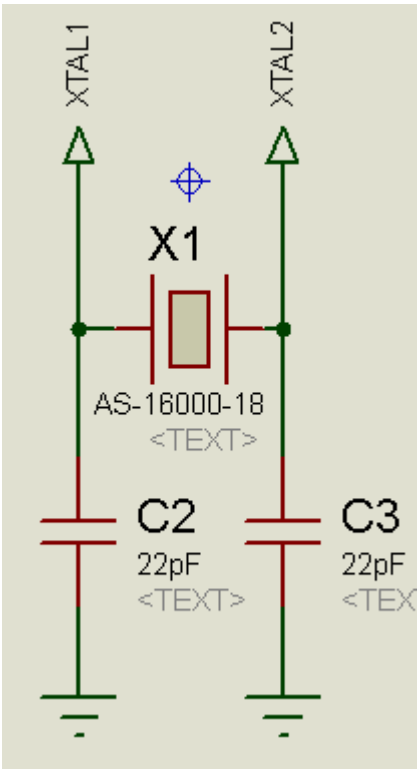


Figure 37 : Schéma de branchement de l'horloge du MCU

Nous avons donc le branchement final du MCU (voir figure 38)

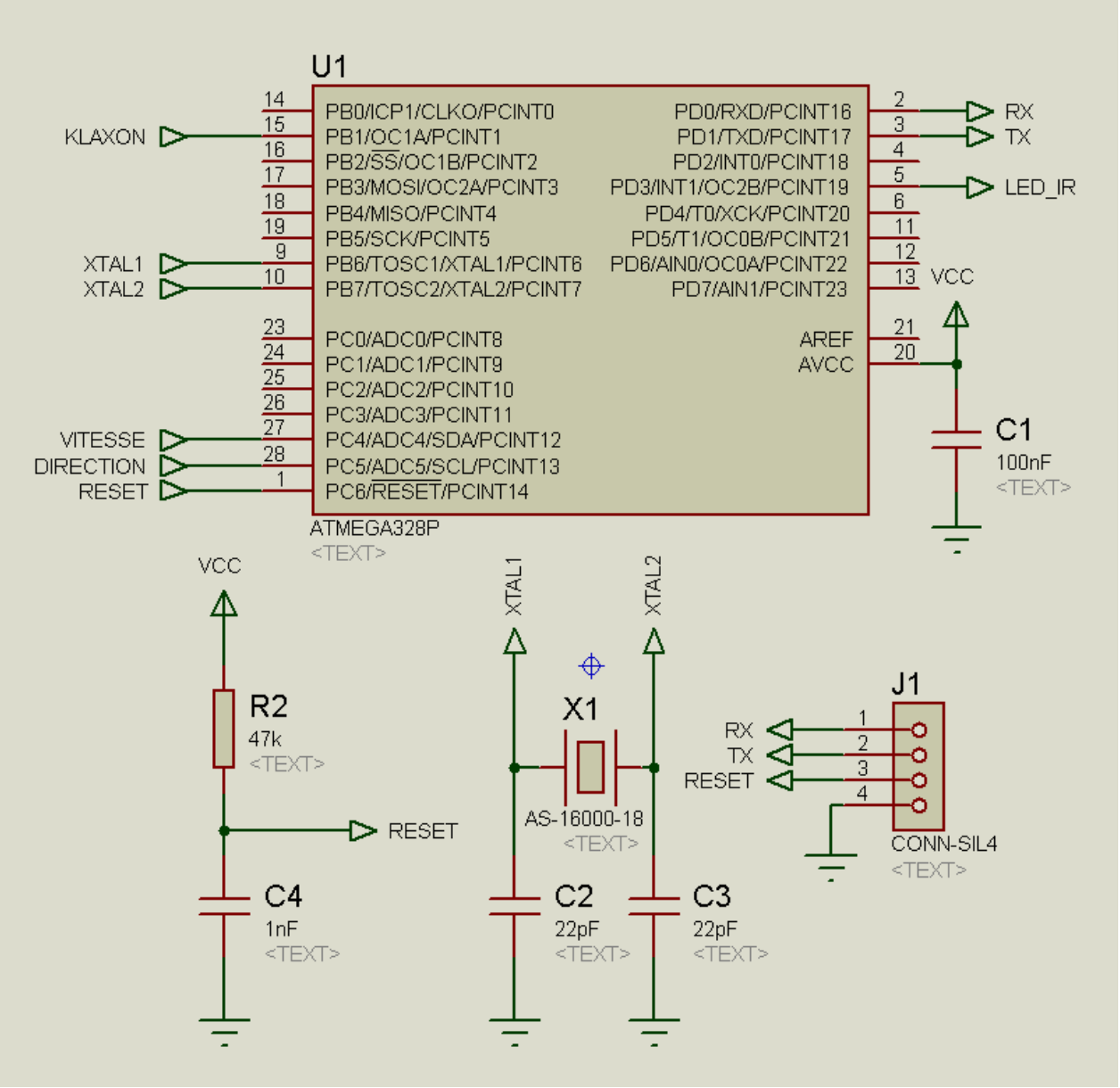


Figure 38 : Schéma du montage du MCU et des périphériques

Référence du paragraphe : CDT_EMTT_PUISSANCE

Rédacteur : Mathéo GRILLET

Relecteur : Mathis BROUSSE et Clément CACHO

Exigences client vérifiées : EXIG_EMTT_PUISSANCE

Compétences GEII : C1-21,22,23,24,25,26

Pour cette partie nous utiliserons les notations présentes sur la figure 39. Nous souhaitons une intensité parcourant les LEDs infrarouges supérieure à 200 mA. La documentation technique des LEDs ne nous permet pas de dépasser 300 mA (seuil de destruction de composant). On prendra donc une valeur de courant intermédiaire afin de limiter le coût des résistances lors de la normalisation de ces dernières. On fixe le courant IL (courant passant dans les LEDs) à 250mA.

Pour cela nous allons définir le courant de saturation du transistor (courant qui traverse la résistance Rb nous l'appelerons Ib).

Nous voulons donc:

$$Ib > \frac{Ic}{\beta}$$

$$\text{avec } \beta = 60 \; et \; Ic = 0.250 \; A <$$

$$Ib > 4.17 \; mA$$

Nous pouvons donc déterminer la valeur de la résistance Rb. On a donc :

$$Rb < \frac{Ub}{Ib}$$

avec Ub la tension aux bornes de la résistance Ub = 3.7 V et Ib = 4.17 mA

$$Rb < 888 \; \Omega$$

On normalise la valeur à 680 Ω de la série E24. Le transistor sera donc saturé.

Nous allons désormais déterminer la valeur de la résistance RL.

Nous avons entre le Vcc et la masse d'une valeur de 5V. La tension Ut aux bornes du transistor est de 0.7V. Les tensions Us de seuil des LEDs est de 1.4V. Par une loi des mailles nous pouvons déterminer la tension Url aux bornes de la résistance RL.

$$Url = Vcc - Ut - 2 \times Us$$

$$Url = 1.5 \text{ V}$$

Par la mise en application de la loi d'ohm nous allons dimensionner la résistance RL.

$$RL = \frac{Url}{Ic}$$

Avec Ic = 0.250 A et Url = 1.5 V

$$Rl = 6 \; \Omega$$

On normalise la valeur à 5.1 Ω de la série E24. Le courant sera donc égal à 0.274 A (min = 0.2514 A et max = 0.289 A) Les LEDs résistent donc et nous répondrons au cahier des charges .

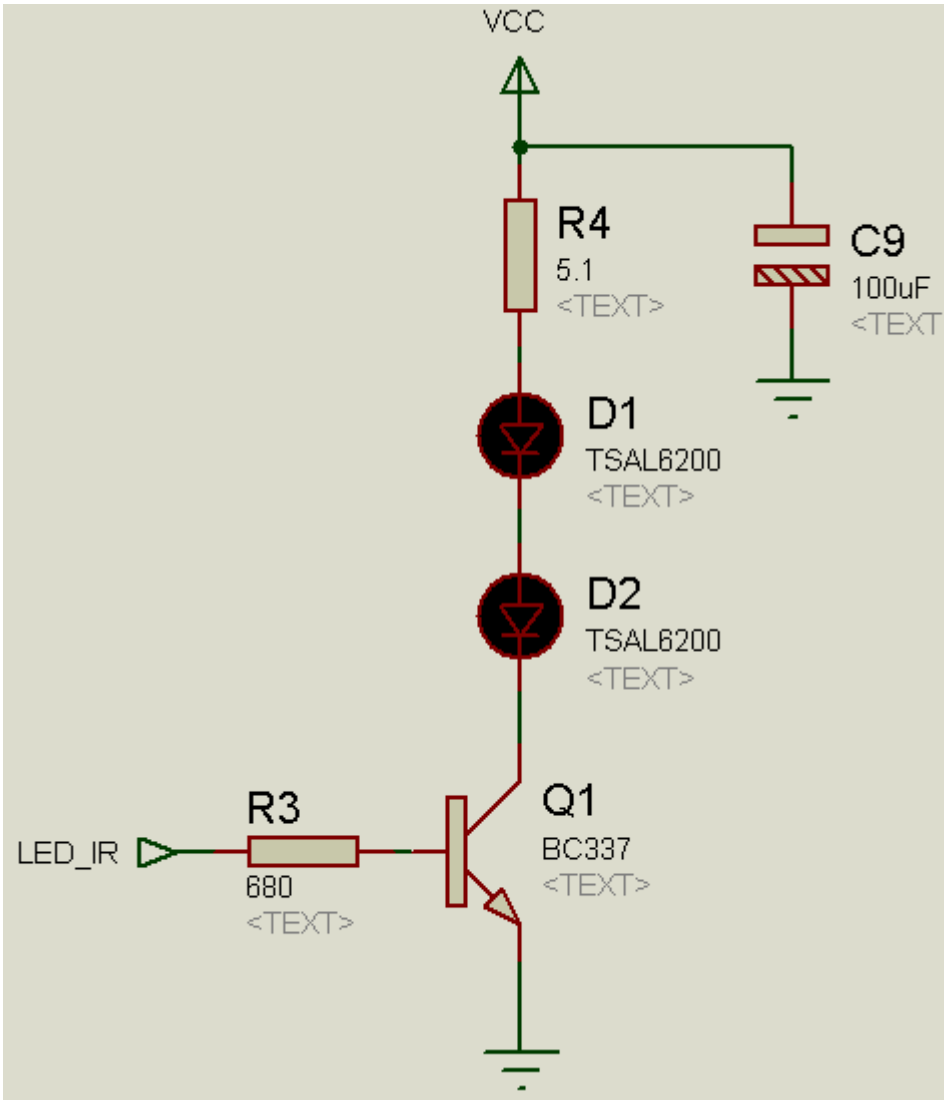


Figure 39 : Schéma du montage des LEDs infra rouge

Référence du paragraphe : CDT_EMTT_INDICATEUR

Rédacteur : Mathéo GRILLET

Relecteur : Mathis BROUSSE et Clément CACHO

Exigences client vérifiées : EXIG_EMTT_INDICATEUR

Compétences GEII : C1-21,22,23,24,25,26

Le cahier des charges nous demande une LED verte qui permet d'indiquer si le montage est mis sous tension.

Ici, on exige une intensité de 50 mcd +/-20%. Nous allons donc utiliser une résistance afin de gérer cette demande.

Pour cela nous devons donc calculer la valeur de cette résistance.

Conformément à la ressource numéro 7 du GEII, nous allons utiliser la formule :

$R = \frac{U_R}{I_R} = \frac{V_{CC} - V_f - V_{OL}}{I_R}$ avec $I_R = I_f$

Pour cela nous réalisons un produit en croix à partir d'une équivalence intensité lumineuse/intensité électrique fournie dans la datasheet.

Nous avons donc : 150 mCd → 20mA

En réalisant un produit en croix nous obtenons : 50 mCd → 6.7mA

Nous avons donc besoin d'un courant $I_f = 6.7mA$.

Dans la datasheet est également fourni une courbe du courant en fonction de la tension V_f (Figure 40)

On obtient :

$V_f = 1.96V$

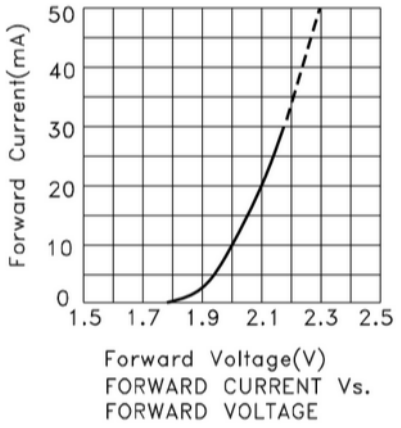


Figure 40 : Courbe du courant en fonction de la tension.

On calcule maintenant R:

$$R = \frac{U_R}{I_R} = \frac{V_{CC} - V_f}{I_R}$$

Application numérique

$$R = \frac{5-1,96}{6,7*10^{-3}} = 454\Omega$$

Nous obtenons donc une résistance de 454Ω.

La résistance choisie est donc une résistance de 470Ω série E6 (+/-10%) .

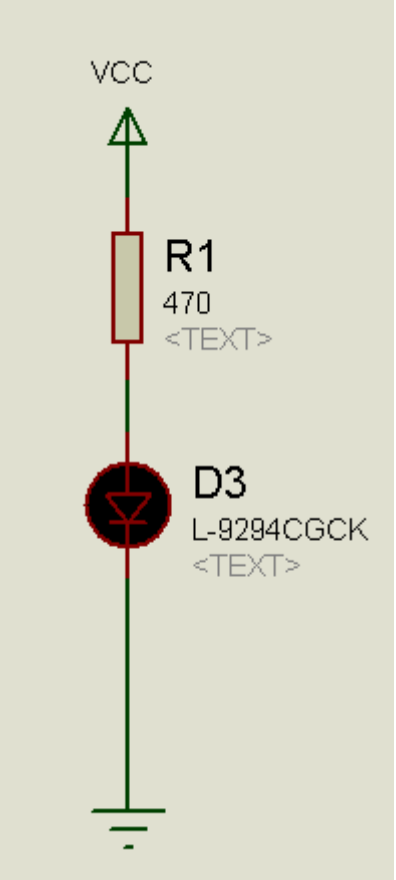


Figure 41: Schéma électrique de la LED

Référence du paragraphe : CDT_EMTT_ENERGIE

Rédacteur : Mathis BROUSSE et CACHO Clément

Relecteur : Mathéo GRILLET

Exigences client vérifiées :EXIG_EMTT_ENERGIE

Compétences GEII : C1-21,22,23,24,25,26

Concernant le bloc énergie nous pouvons faire le choix de 2 composants pour le choix du régulateur. En effet, nous devons choisir quel régulateur linéaire et quel accumulateur utiliser. Dans un premier temps nous pourrons détailler le choix de notre régulateur. L'un des deux régulateurs proposés est un régulateur variable et l'autre un régulateur linéaire fixe. Or nous n'avons besoin que d'un régulateur linéaire fixe, nous pouvons donc orienter notre choix vers le régulateur LM78L05ACZ. Nous pouvons donc par la suite vérifier si les caractéristiques de ce régulateur sont compatibles avec notre utilisation. Concernant la tension d'alimentation, elle doit être comprise entre 7 et 30V, or nous alimentons le circuit avec un accumulateur LIPO 2S de tension nominale 7.4V donc cette caractéristique est vérifiée. La tension de sortie doit être de 5V pour notre circuit,

cette valeur est proposée par les constructeurs de ce régulateur. Concernant le courant de sortie, nous avons besoin d'un courant de 250 mA sur une période totale du régulateur pour alimenter le transistor. Or le transistor ne consomme du courant que sur environ 1⁄3 de la période du régulateur, afin de générer un protocole NEC correspondant à l'exigence (EXIG_EMTT_PUISSANCE). Nous avons donc besoin d'un courant moyen de 83.3mA. Le régulateur délivre au maximum 100mA, cette caractéristique est donc vérifiée. Toutes les caractéristiques électriques du régulateur linéaire sont compatibles avec notre utilisation, nous pouvons donc utiliser celui-ci. D'après la datasheet du régulateur linéaire, nous devrons utiliser des condensateurs de découplage afin d'assurer le bon fonctionnement du régulateur linéaire. Ces condensateurs ont pour valeur 0.1µF et 0.33µF. (voir figure 42)

Figure 42 : Schéma électrique du branchement du régulateur linéaire

Par soucis de logistique nous utiliserons un condensateur 0.47µF à la place du 0.33µF, cette valeur étant supérieur à la valeur initiale, elle est utilisable.

Nous obtenons donc le montage suivant : (voir figure 43)

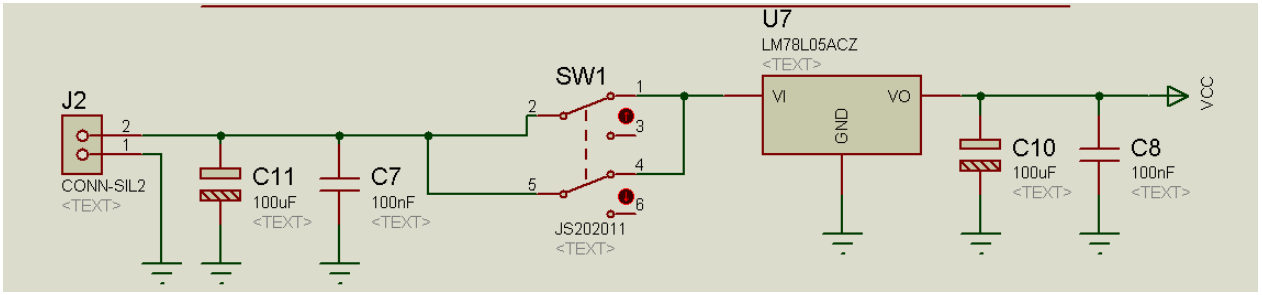


Figure 43 : Montage des condensateurs de découplage du régulateur linéaire

Nous avons par la suite réuni les courants consommés par les différents composants de la carte afin de dimensionner l'accumulateur.

Nom du composant (référence)	Courant d'alimentation	Explication du calcul
MCU (ATMEGA328P)	9 mA	Extrait datasheet (p 303)
LED verte	6.66 mA	Datasheet et rapport de proportionnalité
LED infrarouge	83.3 mA	Courant de 250mA sur une période, consommation sur ⅓ d'une période
Régulateur linéaire (LM78L05ACZ)	0	Négligeable
Potentiomètre linéaire	0.5mA	Tension Vcc = 5V ; R = 10kΩ
Potentiomètre circulaire	0.5mA	Tension Vcc = 5V ; R = 10kΩ
Courant total	99.96mA	Somme des courants

Concernant la LED verte nous avons calculé le courant nécessaire afin de faire briller la LED à une intensité de 20mCd afin qu'elle soit visible facilement à plus d'un mètre. D'après la datasheet de la LED verte, nous savons que pour un courant de 20 mA la LED brille à 150 mCd. Nous savons par approximation que l'évolution de l'intensité lumineuse en fonction du courant est linéaire. Nous pouvons donc calculer par proportionnalité le courant de la LED I = 6.66mA. Concernant les LED infrarouges nous avons besoin d'un courant supérieur à 200mA d'après les renseignements donnés par le cahier des charges, or d'après la datasheet le courant maximal pour une LED infrarouge est de 300mA. Nous avons donc choisi d'utiliser une valeur moyenne de 250mA. Or la LED infrarouge consomme du courant que sur ⅓ de période donc nous avons un courant de 83.3mA. Finalement nous avons calculé le courant total nécessaire à la carte. Nous obtenons I_{tot} = 99.96mA.

Nous avons par la suite calculé l'énergie nécessaire à l'aide de la relation suivante ; $E = I_{tot} * t$.

Avec t = 1 heure (durée de fonctionnement donnée par le cahier des charges)

donc E = 99.96*1

E = 99.96 mAh

Cependant on ne peut pas acheter un accumulateur avec la valeur parfaite pour des raisons de sécurité car on ne décharge pas totalement un accumulateur. De plus, au fil du temps l'accumulateur perd de la capacité dû au cycle charge et décharge. Nous prenons alors selon la ressource numéro 8 du GEII une marge de 20% pour calculer la valeur finale de l'énergie nécessaire

ainsi E = 99.96*1.2

E = 119.95 mA/h

Nous avions le choix parmi trois accumulateurs (350 mAh, 500 mAh, 1000 mAh). Les trois accumulateurs étant compatibles, nous avons choisi finalement par un critère économique. Nous allons donc prendre l'accumulateur LiPo 2S 350mAh car il est le moins cher des trois.

Cependant, sur l'étage des LED infrarouges, nous avons un problème d'alimentation dû à la résistivité de la piste en cuivre qui relie le VCC à cet étage. La tension qui arrive sur cet étage a donc une légère perte qui pourrait perturber les LED. Elles pourraient alors ne plus marcher correctement. Afin de résoudre ce problème, nous pouvons ajouter un condensateur de découplage entre le VCC et la masse de cet étage. Cela permettrait de lisser la tension arrivant sur cet étage et que les pertes de la tension d'alimentation soit inférieures à 1% de celle-ci. Afin de dimensionner ce condensateur, nous devons utiliser la formule suivante (d'après la ressource 25 du département GEII) :

$C > I \cdot \frac{\Delta t}{\Delta Vc}$ avec I qui est le courant consommé par les LED infrarouges lors d'une impulsion,

Δt qui est l'inverse de la fréquence de clignotement des LED,

ΔVc qui est la perte de tension (1% de Vcc).

AN : $C > 83.3*10^{-3}*\frac{\frac{1}{38000}}{0.01*5}$

$C > 43.8\mu F$

Nous devons normaliser ce condensateur et nous allons prendre une série E3 avec une tolérance de 20%. Le condensateur que nous allons utiliser sera : $C_{utilisé} = 100\mu F$

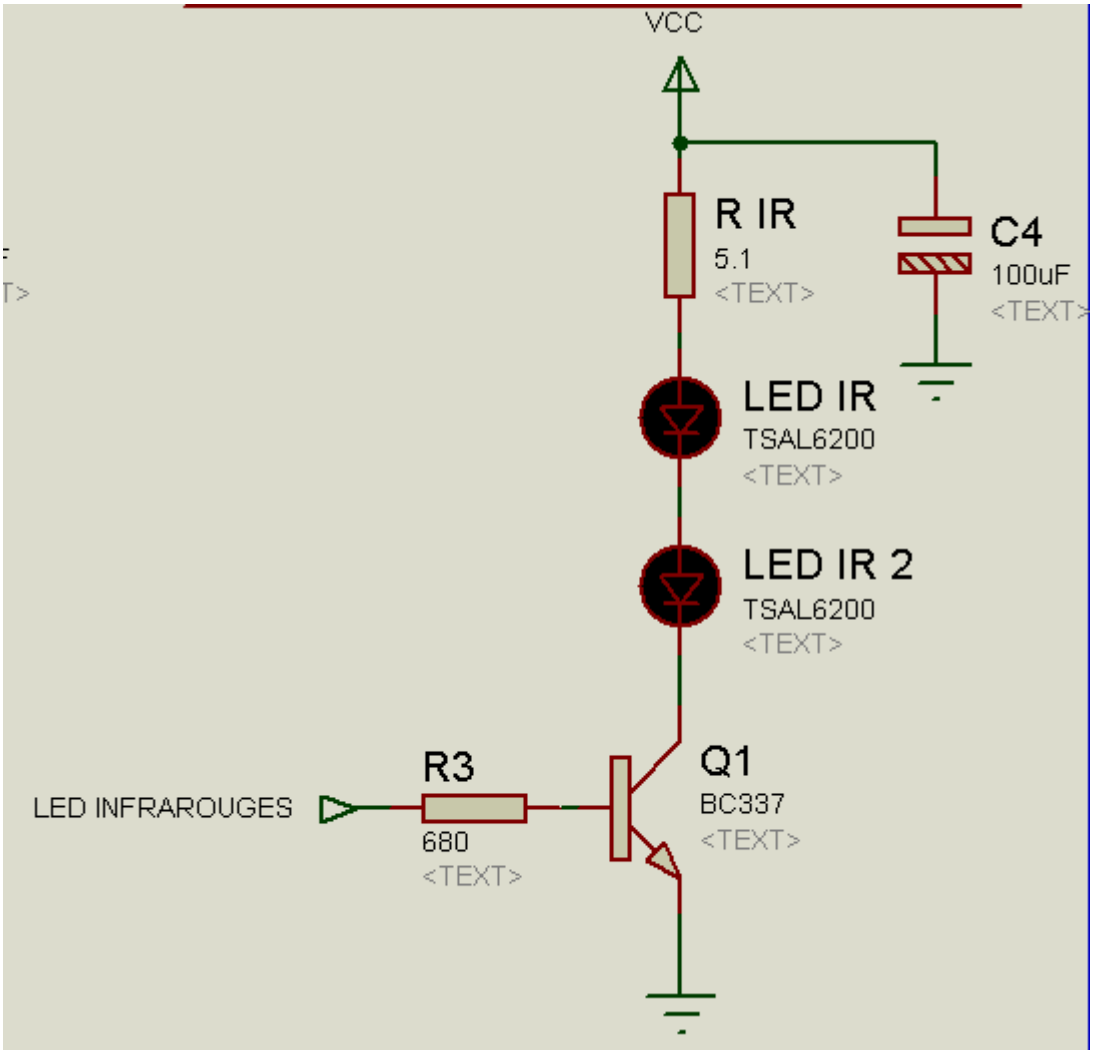


Figure 44 : Montage du condensateur de découplage de l'étage à LED infrarouges

De la même manière, nous devrons utiliser un condensateur de découplage sur le liaison entre le MCU et l'alimentation VCC. Nous utilisons la même relation permettant de calculer la valeur de la capacité du condensateur. Nous savons que $\Delta t = 1/16\ 000\ 000$s et I = 9.5mA. D'après l'application numérique nous obtenons C > 12nF. Nous pourrons donc utiliser un condensateur de $C_{utilisé}$ = 100nF d'après la normalisation de la valeur précédente.

Nous obtenons le montage suivant (voir figure 45)

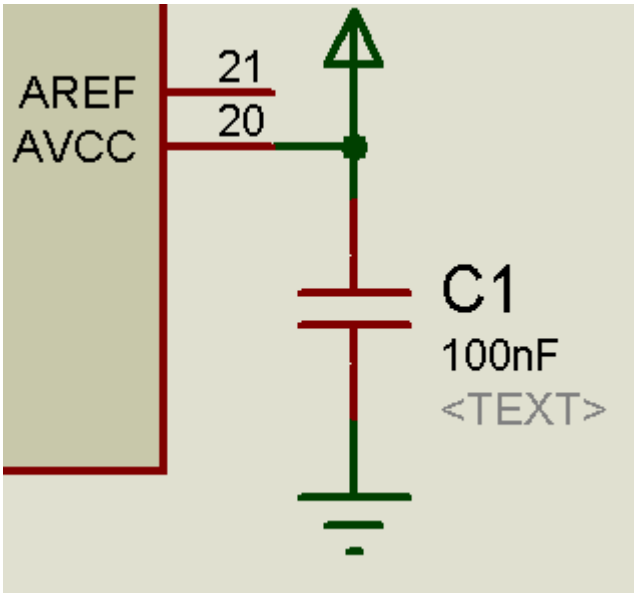


Figure 45 : Montage du condensateur de découplage du MCU

Référence du paragraphe : CDT_EMTT_SCHEMA

Rédacteur : Clément Cacho Mathéo Grillet Mathis Brousse

Relecteur : Clément Cacho Mathéo Grillet Mathis Brousse

Compétences GEII : C1-21,22,23,24,25,26

Voir ci-dessous le schéma électrique suite à la conception détaillé du produit complet.

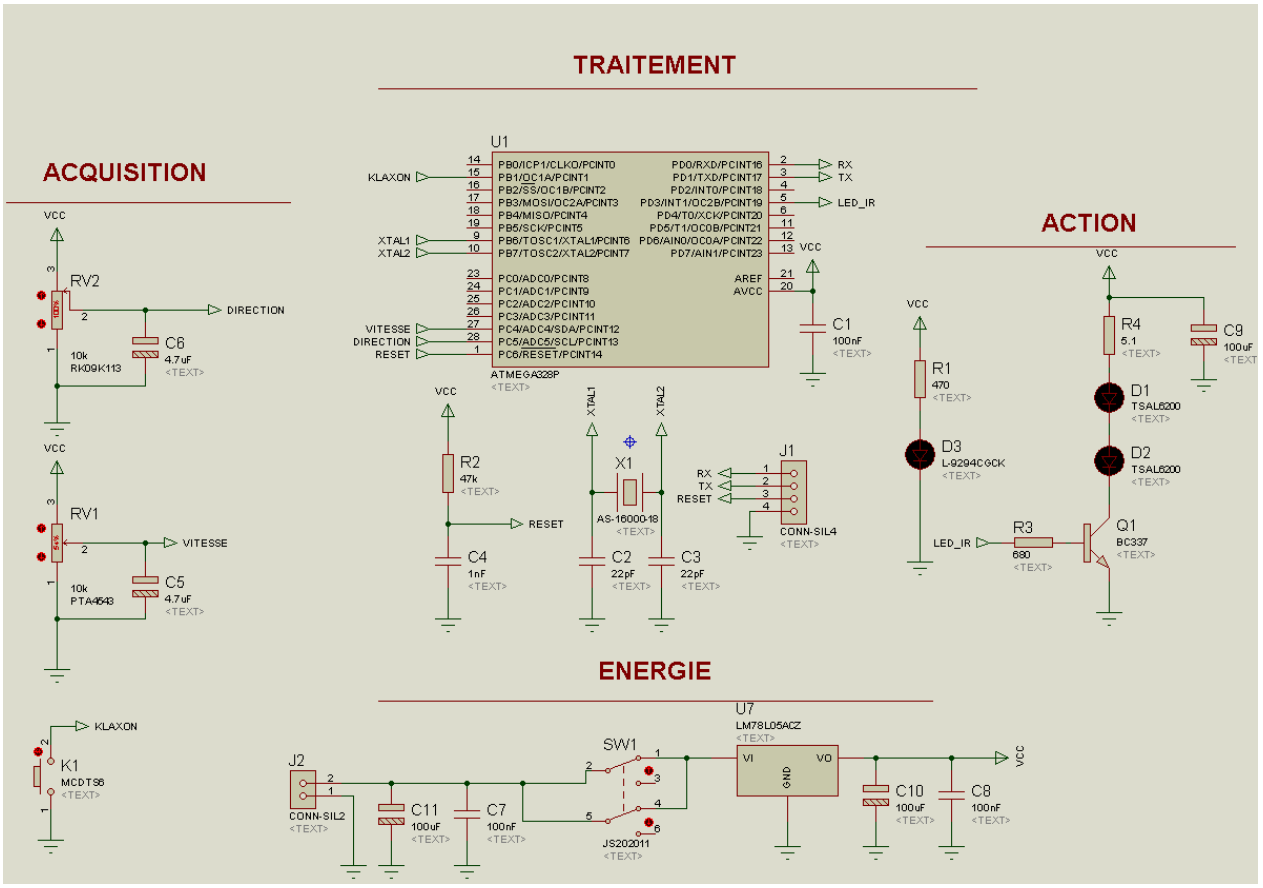


Figure 46 : schéma électrique final de l'émetteur

3.2.2 Récepteur

Référence du paragraphe : CDT_RCPT_TRAITEMENT

Rédacteur : TISSOT Nicolas/GIBELIN Thomas

Relecteur : Ylhan Delannoy Cordeau Maxence

Exigences client vérifiées :
EXIG_RCPT_TRAITEMENT/EXIG_RCPT_SECURITE/EXIG_RCPT_RETENTISSEMENT

Compétences GEII : C1-21,22,23,24,25,26

Dans cette partie, nous devons dimensionner le condensateur et la résistance permettant de générer le signal RESET.

Nous avons extrait de la datasheet la valeur du condensateur : C2 = 1nF. Ce dernier sera non polarisé car sa valeur est inférieur à 1µF. De la même manière nous avons extrait la valeur de V_{IL} = $0.1V_{cc}$ ainsi que $t_{reset} > 2.5\mu s$.

Pour sélectionner ces valeurs nous nous sommes placés dans le cas où nous avons le pire des composants. Or nous avons la relation de la tension aux bornes du condensateur :

$\mathbf{V_c(t) = V_{c\infty} - (V_{c\infty} - V_{cini})e^{-t/\tau}}$ avec $\mathbf{V_{c\infty} = V_{cc}}$; $\mathbf{V_{cini} = 0}$

Or nous savons que pour un temps t_{reset} la tension $\mathbf{V_c(t_{reset}) = V_{IL} = 0.1V_{cc}}$

Nous pouvons donc réécrire notre relation : $\mathbf{V_c(t) = V_{cc}(1 - e^{-t/\tau})}$

Donc nous pouvons dimensionner la résistance R par l'équation suivante :

$V_c(t_{reset}) = V_{cc}(1 - e^{-treset/\tau}) < V_{IL}$

$\leftrightarrow 1 - e^{-treset/RC} < V_{IL}/V_{cc}$

$\leftrightarrow e^{-treset/RC} > 1-V_{IL}/V_{cc}$

$\leftrightarrow -treset/RC > \ln(1-V_{IL}/V_{cc})$

$\leftrightarrow R > -treset/\ln(1-V_{IL}/V_{cc})*C$

Application numérique:

R >-25*10^-6 /ln(1-0.5/5)*1+10^-9

$\leftrightarrow$**R>23728Ω**

Après la valeur calculée, nous cherchons la valeur normalisée de la résistance. N'ayant pas de contrainte de tolérance, nous choisissons la série la moins chère avec une valeur la plus proche et supérieure à la valeur calculée.

Ainsi, nous choisissons une résistance de 47000Ω E3 (+/-20%).

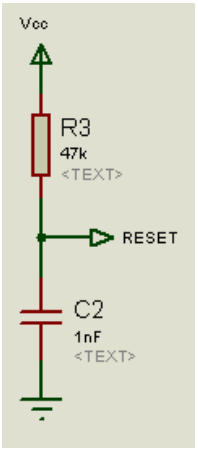


Figure 47 :Schéma électrique de l'étage RESET du MCU

Pour permettre de vérifier $t_{reset} > 2.5\mu s$, on isole t_{reset} et on le calcule avec R= 47000Ω.

$V_c(t_{reset}) = V_{cc}(1 - e^{-treset/\tau}) = V_{IL}$

$\leftrightarrow 1 - e^{-treset/RC} = V_{IL}/V_{cc}$

$\leftrightarrow e^{-treset/RC} = 1-V_{IL}/V_{cc}$

$\leftrightarrow -treset/RC = \ln(1-V_{IL}/V_{cc})$

$\leftrightarrow treset = -\ln(1-V_{IL}/V_{cc})*RC$

Application numérique:

$treset = -\ln(1-0.5/5)*47000*1*10^{-9}$

treset = 4.95µs > 2.5µs

On peut donc valider la valeur de R choisit puisque treset est bien supérieure à 2.5µs.

Pour ce qui est de la tension du Microcontrôleur, utilisant un **BEC** (Battery Elimination Circuit) qui délivre une tension au récepteur de **5V** et le microcontrôleur supportant une tension de 1.8V à 5.5V, celui-ci reçoit donc une tension de 5V.

Pour ce qui est du courant du Microcontrôleur, dans la datasheet nous trouvons une courbe du courant en fonction de la fréquence (figure 45) pour une tension de 5V dans le MCU. L'ATMEGA 328P ayant une fréquence d'horloge de 16MHz, cela nous permet de déterminer un courant parcourant le MCU de 9.5 mA.

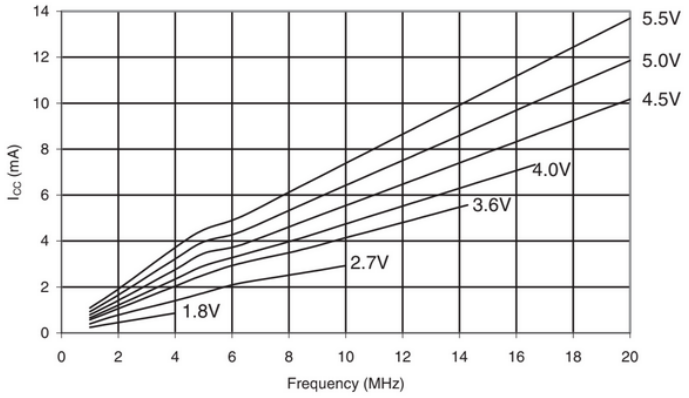


Figure 48 :Courbe de la tension du microcontrôleur

Grâce à la datasheet du microcontroleur ainsi qu'au schéma électrique du jeu Simon, nous avons pu déterminer les entrées et les sorties reliées au MCU.

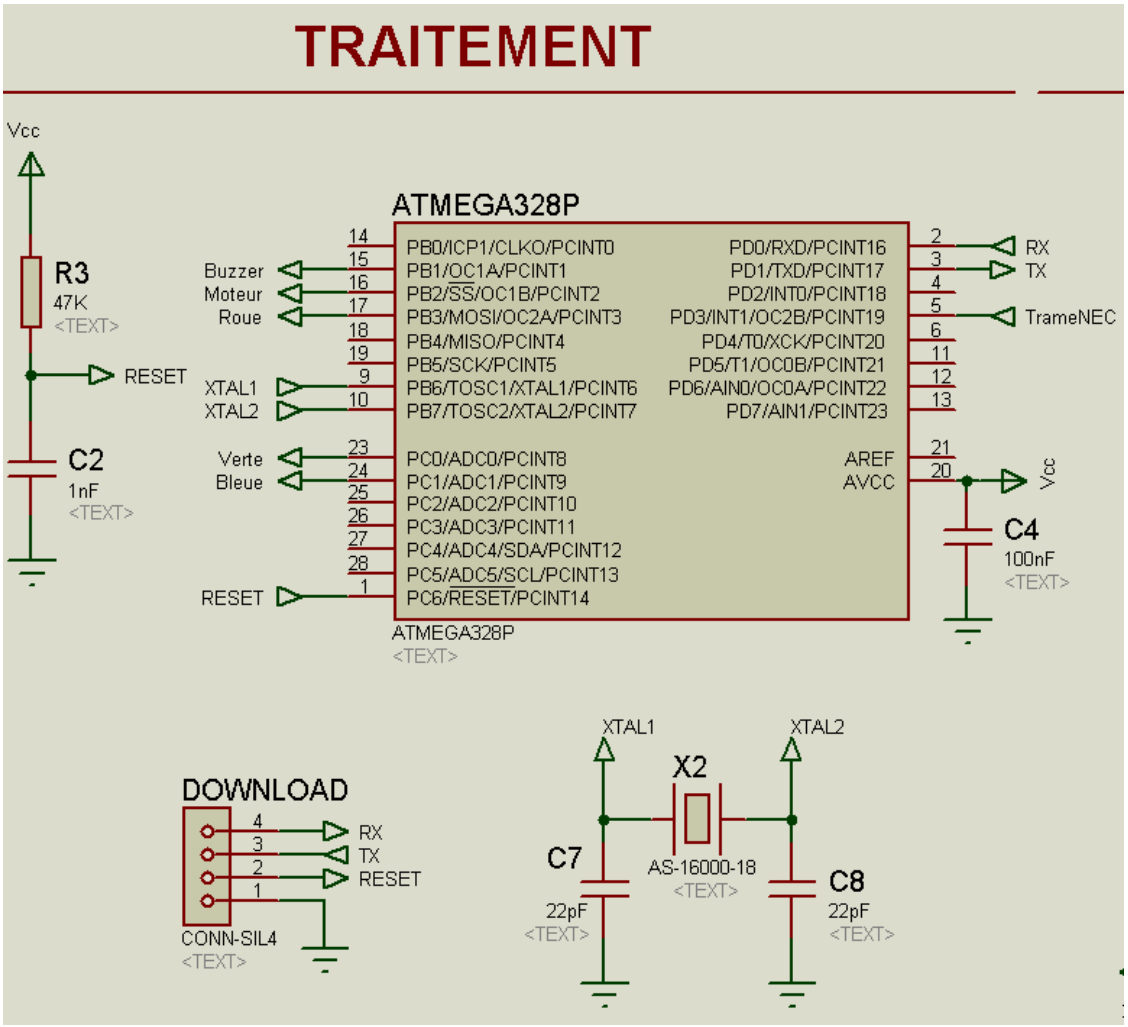


Figure 49 :Schéma électrique du bloc Traitement

Référence du paragraphe : CDT_RCPT_ENERGIE

Rédacteur : TISSOT Nicolas/GIBELIN Thomas

Relecteur : Ylhan Delannoy Cordeau Maxence

Exigences client vérifiées : EXIG_RCPT_ENERGIE/EXIG_RCPT_INTERRUPTEUR

Compétences GEII : C1-21,22,23,24,25,26

Dans le cahier des charges, il est indiqué que nous utilisons un accumulateur Lipo 2S.

On cherche à trouver tout d'abord l'intensité du courant totale I correspondant à la somme des intensités nécessaire à chaque composant :

Nom	Nbr	Courant (en A)
Buzzer	1	2 mA
ATMEGA 328P	1	9.5 mA
Moteur Brushless	1	2500 mA
Servomoteur	1	7.4 mA
Diode verte	1	6.7 mA
Diode bleu	1	4 mA
	Somme	2529.6 mA

On trouve I= 2529.6 mA.

D'après le cahier des charges, l'un accumulateur LiPo 2S, doit assurer une autonomie minimum de fonctionnement de 15 min.

On utilise la formule : **E=I*Δt**, pour pouvoir trouver la capacité de l'accumulateur et nous ajoutons 20% comme indiqué dans le HTUT 8 : "Comment dimensionner un accumulateur ?".

A.N : E= 2529.6 * 0.25*1,20 = 758.88 mAh

Nous avons donc besoin d'un accumulateur Lipo 2S de 1000 mAh minimum.

Nous avons retenue un accumulateur de 1000 mAh

On décide donc de calculer l'autonomie du récepteur avec cette valeur.

$\Delta t = \frac{E}{I}$

A.N : $\Delta t = \frac{1000*1.2}{2529.6} = 0.474h = 28.44$ min

Cela nous permet donc d'assurer une autonomie de 28.44 min.

Référence du paragraphe : CDT_RCPT_CAPTEUR

Rédacteur : Ylhan Delannoy Maxence Cordeau

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Exigences client vérifiées : EXIG_RCPT_CAPTEUR

Compétences GEII : C1-21,22,23,24,25,26

Dans ce bloc nous expliquons plus en détails la manière dont nous mettons en place le récepteur infrarouge permettant de récupérer les informations essentielles envoyées par l'émetteur permettant de contrôler le Kart à hélice.

Tout d'abord lors de notre analyse des différents récepteurs proposés nous finissons par choisir le TSOP4438. Précédemment nous avions indiqué dans la rédaction du dossier de conception préliminaire que la raison était dû au boîtier cependant la réelle raison est que la zone de réception de ce boîtier est beaucoup plus grande que le deuxième récepteur proposé. Ce qui nous permet de pouvoir récupérer la trame envoyée avec plus de facilité sans avoir d'interruption.

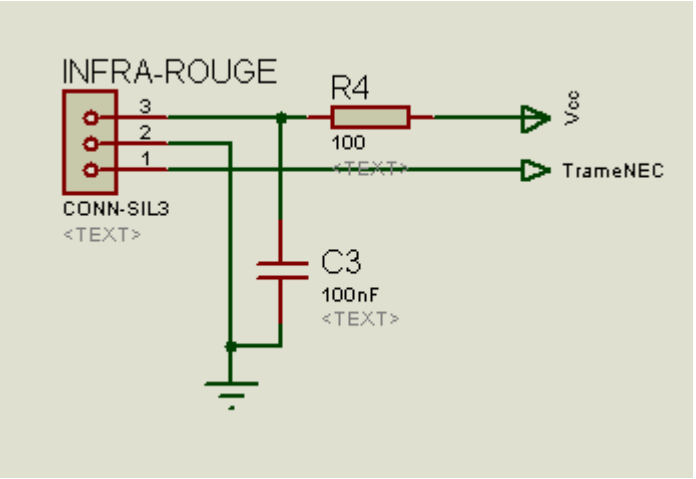


Figure 50 : Schéma électrique de l'étage récepteur infra-rouge

Ce capteur a pour objectif de recevoir les trames NEC reçues depuis l'émetteur. Ce composant est doté de 3 broches, V_{CC} la tension d'alimentation d'entrée V_{EE} la tension d'alimentation de sortie et V_{OUT} la tension contenant les informations du signal reçu. En ce qui concerne le branchement de ce capteur, il est conseillé d'après la datasheet de l'associer à un condensateur et à une résistance. Le condensateur de découplage permettra de lisser le signal afin d'avoir le moins de perturbations possibles au niveau du MCU. Voici un schéma de représentation d'un signal lissé grâce au condensateur de découplage.

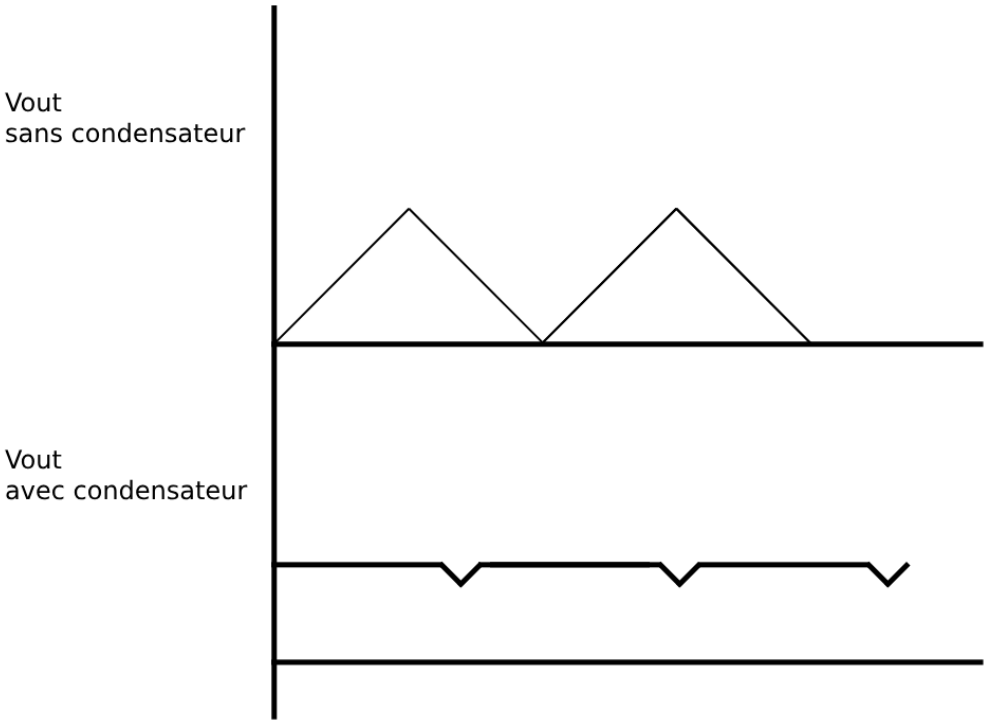


Figure 51 : schéma de représentation d'un signal brut et d'un signal lissé par un condensateur

Pour cela nous réalisons le calcul du condensateur avec la formule ci-dessous:

$I = C * \Delta V/\Delta T$

donc $C = I * \Delta T/\Delta V$

soit $I * 1/\Delta V * f$ car $\Delta T = 1/f$

donc $C = 5 * 10\text{^} - 3 * 1/38 * 10\text{^}3 * 0.01 * 5$ avec $C > 2.63 * 10\text{^} - 6$

Nous réalisons ce calcul permettant de connaître la valeur de la capacité nécessaire cependant après analyse de la datasheet nous relevons une valeur conseillée de 0,1µF.

Par précaution nous préférons utiliser celle-ci. Étant donné que les informations données en datasheet ont été vérifiées à de nombreuses reprises lors de différents tests, nous décidons de l'utiliser.

Une résistance de 100Ω est donc nécessaire.

En ce qui concerne les valeurs de normalisations de la résistance et du condensateur, nous choisissons un condensateur de 100nF non polarisé et pour ce qui est de la valeur de la résistance de 100 ohm de série E3 à +/- 20%

Dès lors nous procédons au dérisquage et réalisons le montage donné en Figure 49.

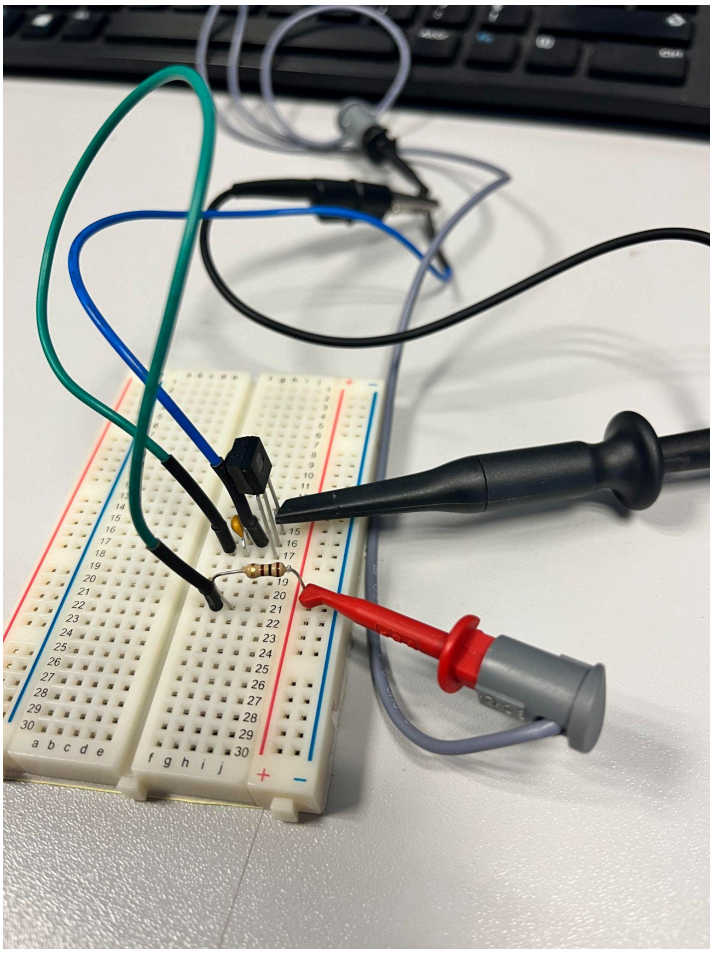


Figure 52: Montage du dérisquage du récepteur infrarouge

Une fois le montage réalisé nous le mettons sous tension afin de vérifier que le signal reçu correspond au signal attendu. Nous branchons donc le capteur infrarouge accompagné d'un condensateur branché en parallèle et d'une résistance sur une plaque de prototypage puis nous mettons une tension de 5V sur un générateur de table afin d'observer le comportement du récepteur lors de la réception d'une trame NEC à l'aide d'un oscilloscope relié au montage avec une sonde. Nous émettons la trame NEC à l'aide d'une télécommande qui nous permet de simuler l'envoie de nos données.

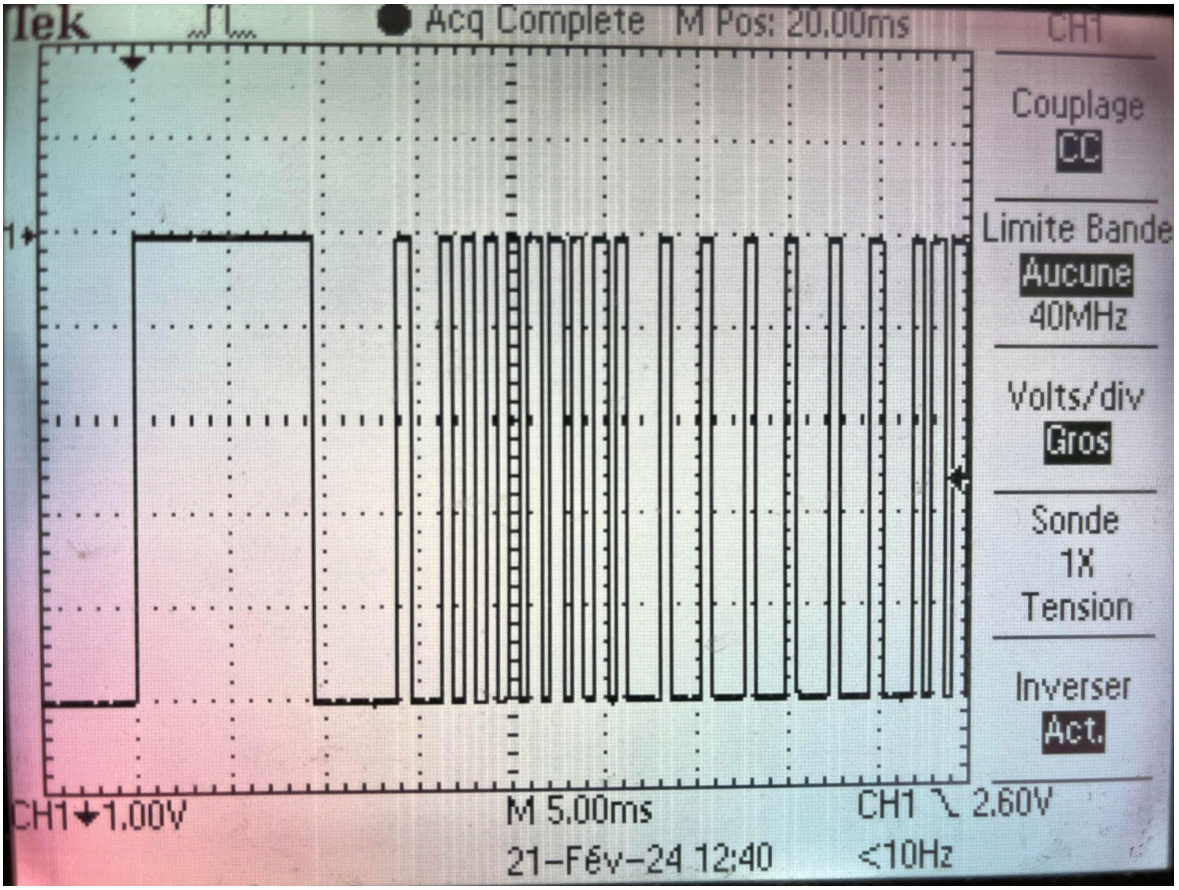


Figure 53 : Réception d'une trame NEC dans le TSOP4438

Nous repérons sur le signal différents pics et remarquons que le récepteur fonctionne en logique inversée, en effet lors de la réception d'un bit de la trame NEC, le signal devrait être fixé à Vcc or nous observons le cas contraire. Nous repérons principalement 3 parties, le premier pic qui correspond à l'en-tête liée au protocole, elle dure 9 ms. Les 8 pics suivants correspondent à l'octet d'adresse composé de l'état du klaxon sur le premier bit et du numéro de l'équipe sur les 7 suivants. Enfin nous voyons un dernier octet composé de 4 bits de direction et de 4 bits de vitesse.

Le cahier des charges spécifiant la réception d'une trame NEC incluant 2 mots respectivement composés du klaxon et de l'adresse puis de la direction et la vitesse, nous le validons à l'aide des informations fournies ci-dessus, l'exigence EXIG_RCPT_CAPTEUR est donc vérifiée et nous sommes conforme au cahier des charges.

Référence du paragraphe : CDT_RCPT_MOTEUR

Rédacteur : Ylhan Delannoy Cordeau Maxence

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Exigences client vérifiées : EXIG_RCPT_MOTEUR

Compétences GEII : C1-21,22,23,24,25,26

Pour contrôler les hélices de notre kart, un moteur est nécessaire. Le moteur doit être autonome sur sa rotation. Nous relevons un moteur brushless, le Turnigy 28-22-CQ 1400Kv qui est connecté à l'aide de 3 broches (l'alimentation "+", l'alimentation "-") ainsi que la broche envoyant le signal PWM qui contrôle la vitesse en fonction du rapport cyclique du signal.

Le moteur brushless est précédé d'un contrôleur BEC qui premièrement permet d'augmenter le courant de sortie et dans un second temps sert aussi à réguler la tension d'entrée pour avoir un moteur stable. La partie traitement est directement impliquée dans le fonctionnement du moteur puisqu'en fonction du signal délivré la vitesse du moteur varie. Nous ajoutons un condensateur de découplage en parallèle du Vcc, afin de limiter les pics de consommation du moteur. Nous prenons une grosse valeur de capacité afin que le condensateur agisse comme un réservoir à courant afin d'absorber la haute consommation de courant au démarrage. Nous choisissons la valeur de condensateur la plus importante disponible d'un point de vue logistique, donc un condensateur polarisé de 470µF.

Caractéristiques du Turnigy 28-22-CQ 1400Kv :

Courant d'entrée maximum	5A
Tension d'entrée	6~9V

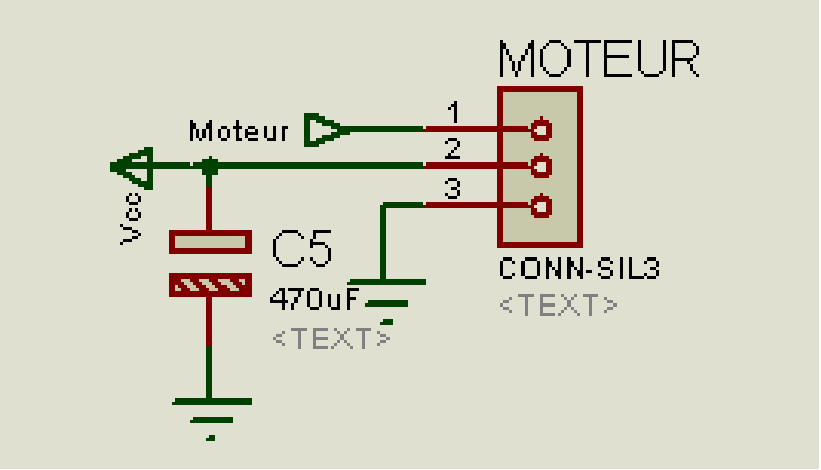


Figure 54 : Schéma du connecteur moteur

Référence du paragraphe : CDT_RCPT_ROUE

Rédacteur : Ylhan Delannoy Cordeau Maxence

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Exigences client vérifiées : EXIG_RCPT_ROUE

Compétences GEII : C1-21,22,23,24,25,26

Nous désirons pouvoir contrôler la direction du Kart à hélices en utilisant un servomoteur.

Ce dernier est chargé de réguler l'angle de direction à l'avant du kart à hélice. Pour ce faire, il est nécessaire de moduler le rapport cyclique du servomoteur : un rapport cyclique faible correspond à un angle de 0°, tandis qu'un rapport cyclique élevé correspond à un angle de 180°, permettant ainsi de diriger le kart vers la gauche (0°) ou vers la droite (180°).

Par ailleurs, le servomoteur est connecté à l'unité de traitement par trois câbles : le premier pour l'alimentation positive (+), le deuxième pour la mise à la terre (-) et le troisième pour le signal PWM, qui permet de contrôler le servomoteur en lui envoyant les instructions appropriées. Nous ajoutons un condensateur de découplage en parallèle du Vcc, afin de limiter les pics de consommation du moteur. Nous prenons une grosse valeur de capacité afin que le condensateur agisse comme un réservoir à courant afin d'absorber la haute consommation de courant au démarrage. Nous choisissons la valeur de condensateur la plus importante disponible d'un point de vue logistique or un condensateur polarisé de 470µF.

En conséquence, nous avons opté pour le servomoteur HS-322HD.

Informations à propos du composants du HS-322HD:

Tension de fonctionnement	4.8 à 6.0V
Courant de fonctionnement maximal	7.4mA

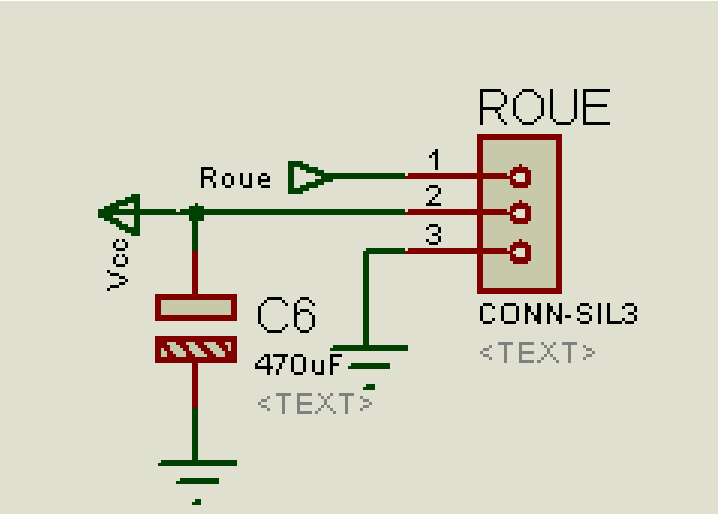


Figure 55 : Schéma du connecteur de la direction

Référence du paragraphe : CDT_RCPT_INDICATEUR

Rédacteur : Ylhan Delannoy Cordeau Maxence

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Exigences client vérifiées : EXIG_RCPT_INDICATEUR

Compétences GEII : C1-21,22,23,24,25,26

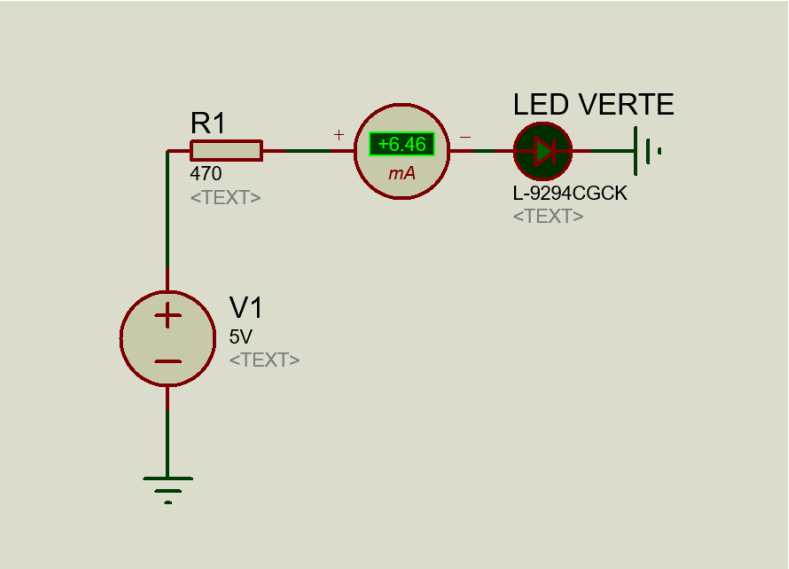


Figure 56: Indicateur mise sous tension

Nous souhaitons avoir un témoin lumineux nous indiquant lorsque le kart est mis sous tension. On nous impose une intensité lumineuse de 50 mcd. Pour cela nous disposons de la datasheet de la LED verte qui nous est mise à disposition. Nous savons grâce à celle-ci que lorsque le courant est de 20mA nous avons une intensité lumineuse de 150 mcd. Donc nous réalisons un produit en croix:

$150\ mcd\ =\ 20mA$

$50\ mcd\ =\ 6,66\ mA$

Après le calcul nous trouvons le courant nécessaire pour avoir 50 mcd qui est de 6,66mA.

Maintenant il nous faut déterminer la valeur de la résistance qui sera mise en amont pour avoir ce courant d'entrée de 6,7mA. Nous utilisons donc la formule ci dessous:

$$R\ =\ \frac{Vcc-(Vcc-Voh)-Vf}{If}$$

Or nous ne sommes pas relié au microcontrôleur donc nous n'avons pas de perte donc

R= $\frac{Vcc-Vf}{If}$ Vcc=5V Vf=1,95 If=6,66

Après calcul nous avons donc R=457Ω

Nous passons donc à la normalisation du composant étant donné que nous voulons une précision de +/-20% nous prenons donc une résistance de série E6+/-10% de 470Ω.

If=Vcc-Vf/R

If=6,48mA

Après réalisation du produit en croix nous avons donc une intensité lumineuse de 48.6mC ce qui rentre dans la marge autorisé qui est de 20%. Car nous avons une erreur relative de 2,8% donc nous respectons l'exigence.

Erreur relative : | (ValeurObtenue - ValeurAttendue) / Valeur attendue | = | (48,6*10^-3 - 50*10^-3) / 50*10^-3 | = 0,028 = 2,8%

Référence du paragraphe : CDT_RCPT_CONNEXION

Rédacteur : Ylhan Delannoy Cordeau Maxence

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Exigences client vérifiées : EXIG_RCPT_CONNEXION

Compétences GEII : C1-21,22,23,24,25,26

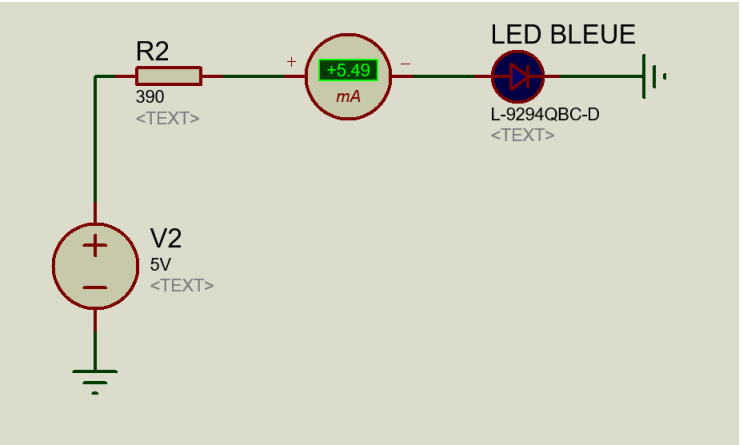


Figure 57: Indicateur mise sous tension

Nous souhaitons avoir un témoin lumineux nous indiquant lorsque le kart reçoit une nouvelle trame NEC. On nous impose une intensité lumineuse de 100 mcd. Pour cela nous disposons de la datasheet de la LED bleue qui nous est mise à disposition. Nous savons grâce à celle-ci que lorsque

le courant est de 20mA nous avons une intensité lumineuse de 500 mcd. Donc nous réalisons un produit en croix :

$500\,mcd = 20mA$

$100\,mcd = 4mA$

Après le calcul nous trouvons le courant nécessaire pour avoir 100 mcd qui est de 4mA.

Maintenant il nous faut déterminer la valeur de la résistance qui sera mise en amont pour avoir ce courant d'entrée de 4mA. Nous utilisons donc la formule ci dessous:

$R = \frac{Vcc-(Vcc-Voh)-Vf}{If}$

Vcc=5V Vf=2.8 Voh= 4.2VIf=6,66

Après calcul nous avons donc R=350Ω

Nous passons donc à la normalisation du composant étant donné que nous voulons une précision de +/-20% nous prenons donc une résistance de série E6 +/-10% de 390Ω.

$If = \frac{Vcc-(Vcc-Voh)-Vf}{R}$

If=3.5mA

Après réalisation du produit en croix nous avons donc une intensité lumineuse de 87.5 mcd ce qui rentre dans la marge autorisé qui est de 20%. Car nous avons une différence de 12.5% donc nous respectons l'exigence.

Erreur relative : (87.5*10^-3 - 100*10^-3) / 100*10^-3 = 0.125 = 12,5%

Référence du paragraphe : CDT_RCPT_KLAXON

Rédacteur : Ylhan Delannoy Cordeau Maxence

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Exigences client vérifiées : EXIG_RCPT_KLAXON

Compétences GEII : C1-21,22,23,24,25,26

Le buzzer MCKPT-G1210-3916 génère d'après la datasheet un signal carré compris entre +6 V et -6V. Nous souhaitons avoir une fréquence de 4kHz à laquelle nous appliquons un filtre passe-bas pour avoir le signal le plus lisse possible.

Pour cela :

La datasheet indique un courant efficace de 2mA et une tension efficace de 6V (puisque le 12Vpp = -6V “+” +6V).

Nous utiliserons le buzzer en sortie d’un filtre passe-bas composé d’un condensateur.

Nous obtenons la résistance R avec la relation U / I telle que $R = U / I = 6 / 2 * 10^{-3} = 3k\Omega$

Pour définir la capacité du condensateur, nous utilisons la formule liée à la fréquence de coupure qui est telle que :

$fc = 1 / 2\pi RC < 4kHz\ avec\ R = 3k\Omega$ (résistance interne du buzzer)

donc $C > 1 / 2\pi * 3000 * 4000 = 1,33 * 10^{-8} = 13,3nF$

donc $C > 13,3nF$

Nous obtenons une valeur de condensateur de 13,3nF, en valeur normalisée nous obtenons 100nF de façon à obtenir un point de coupure qui n’influence pas sur la qualité du signal carré. Plus la valeur de condensateur est élevée plus les basses fréquences peuvent passer, 100nF est donc une valeur correcte pour laisser passer les fréquences que nous souhaitons tout en filtrant les fréquences non désirées.

En couplant le buzzer à un condensateur de 100nF, nous obtenons un signal plus lisse qui permet d’obtenir un signal plus ou moins constant pour éviter les oscillations de fréquence.

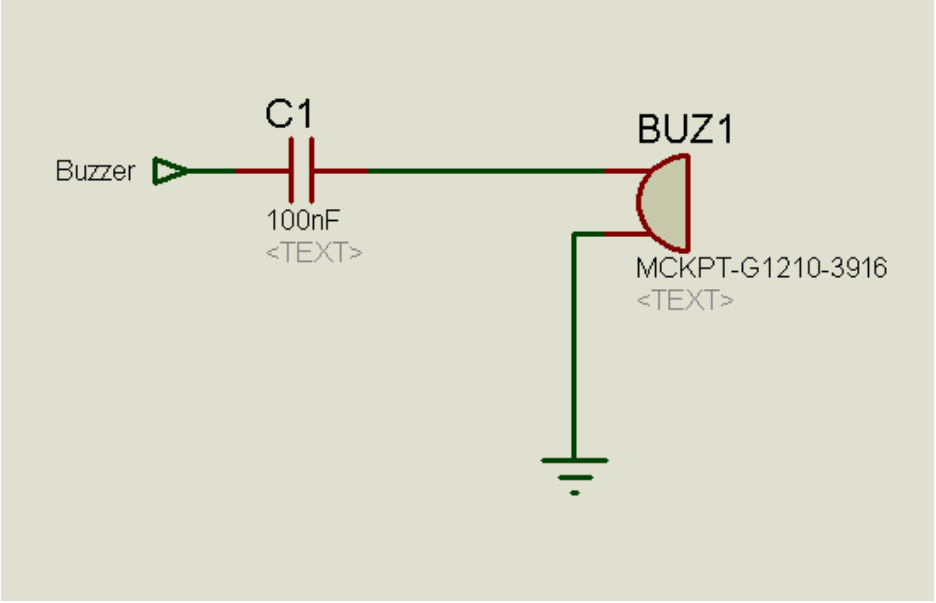


Figure 58 : Schéma du buzzer mis sous tension

En sortie de ce montage nous obtenons un signal carré entre 0 et 5V en PWM avec un rapport cyclique de 50% qui permet de faire vibrer le buzzer à une fréquence 4kHz comme indiqué dans le cahier des charges. Les conditions de conformités étant vérifiées, l'étage correspondant au buzzer est terminé et conforme.

Référence du paragraphe : CDT_EMTT_SCHEMA

Rédacteur : TISSOT Nicolas/GIBELIN Thomas

Relecteur : Ylhan Delannoy Cordeau Maxence

Compétences GEII : C1-21,22,23,24,25,26

Voir ci-dessous le schéma électrique suite à la conception détaillé du produit complet

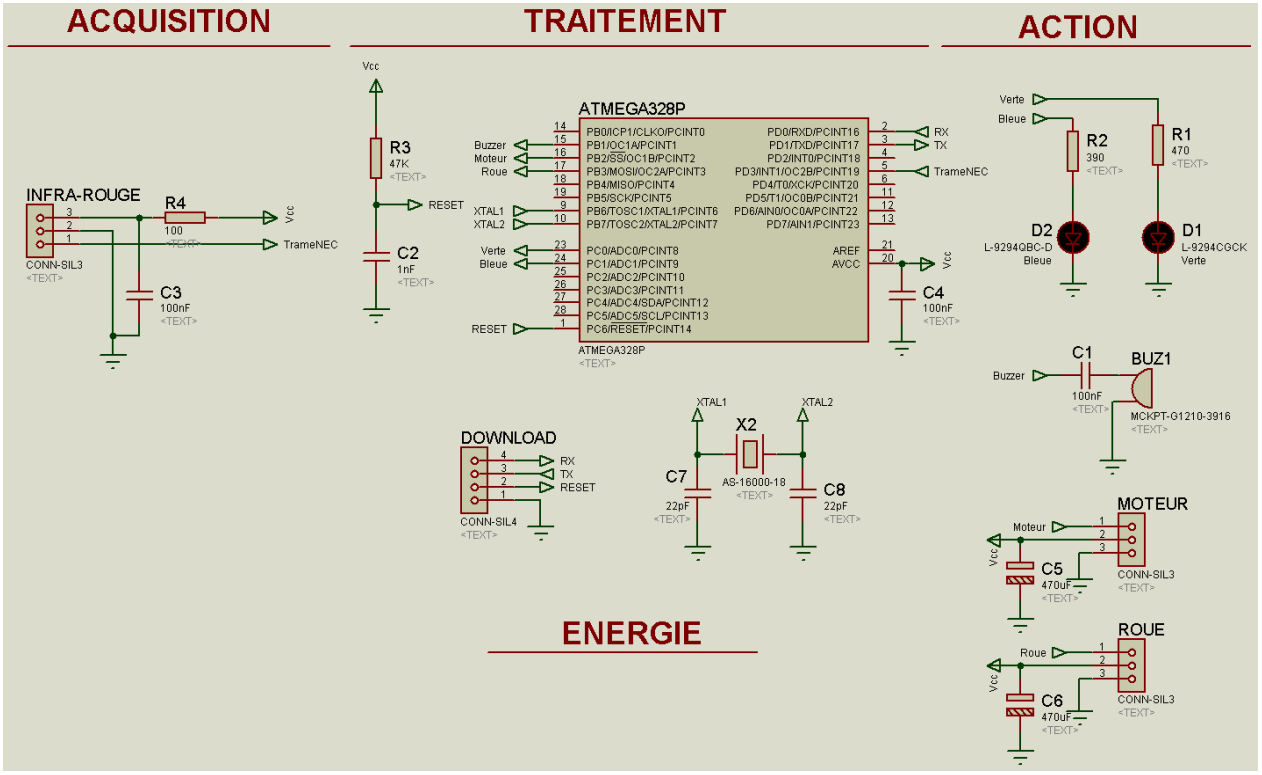


Figure 59 : schéma électrique final du récepteur

3.3 Informatique

3.3.1 Emetteur

Référence du paragraphe : CDT_EMTT

Rédacteur : Ylhan Delannoy Cordeau Maxence

Relecteur : TISSOT Nicolas/GIBELIN Thomas

Compétences GEII : C1-21,22,23,24,25,26

Dans cette partie nous avons mis en place un programme afin de pouvoir transmettre les informations relatives aux commandes de l'utilisateur que nous souhaitons envoyer au Kart à hélice.

Pour ce faire, plusieurs fonctions sont utilisées :

- La fonction AcquerirPotarVitesse récupère la valeur du potentiomètre linéaire qui définit la vitesse du kart. Elle retourne sa valeur qui est un entier compris entre 0 et 1023.

- La fonction AcquerirPotarRotation récupère la valeur du potentiomètre circulaire qui définit la direction du kart. Elle retourne sa valeur qui est un entier compris entre 0 et 1023.

- La fonction AcquerirBoutonKlaxon récupère l'état du bouton correspondant à la commande du klaxon. Elle retourne un booléen de valeur True si le bouton est pressé ou False s'il n'est pas pressé

- La fonction CalculerAdresse calcule l'octet d'adresse, il fusionne les 2 octets de 1 et 7 bits correspondant chacun au klaxon et au numéro d'équipe. Elle contient 2 paramètres klaxon de type booléen et et numeroEquipe de type byte ainsi qu'une variable adresse de type byte elle retourne adresse, étant un octet de 8 bits, le premier étant l'état du klaxon et les 7 suivants le numéro de l'équipe

- La fonction CalculerDonnee calcule l'octet de données, il fusionne les 2 octets de 4 bits correspondant chacun à la vitesse et à la direction du kart. Elle contient 2 variables vitesse et rotation subissant une transformation de 1024 valeurs possibles à seulement 17 pour la vitesse et 16 pour la direction. Nous choisissons 1024 valeurs de façon à éliminer l'imperfection de la division des valeurs par le map de façon à être capable d'atteindre la vitesse maximale qui avec 1023 valeurs est difficilement atteignable et provoque un oscillement de la vitesse la 15 et la 16ème vitesse. Elle retourne un octet de 8 bits contenant d'abord la vitesse puis la direction

- La fonction loop() exécute en boucle le code se trouvant à l'intérieur

  Elle initialise 5 variables :

  - potarVitesse de type int qui stocke la valeur du potentiomètre destiné à la commande de la vitesse donnée par la fonction AcquerirPotarVitesse()

- potarRotation de type int qui stocke la valeur du potentiomètre destiné à la commande de la direction donnée par la fonction AcquerirPotarRotation()

- klaxon de type booléen qui stocke l'état du klaxon destiné à piloter le klaxon du kart donné par la fonction AcquerirBoutonKlaxon()

- adresse de type byte qui stocke l'octet de d'adresse donné par la fonction CalculerAdresse() destiné à être envoyé dans la trame NEC afin de transmettre les informations relatives au klaxon et au numéro d'équipe au récepteur du kart

- donnee de type byte qui stocke l'octet de donnée donné par la fonction CalculerDonnee() destiné à être envoyé dans la trame NEC afin de transmettre les informations relatives à la direction et à la vitesse au récepteur du kart

La boucle se termine par l'envoi de la trame NEC générée par la fonction GenererTrameNEC() qui combine l'octet d'adresse et l'octet de donnée afin de l'envoyer au récepteur.

```
/*****************************************************************/
// Sujet :   Programme de l'émetteur du kart électronique        //
// Auteur :  CORDEAU Maxence, DELANNOY Ylhan                     //
// Date :    13/03/2024                                          //
// Version : 1.0                                                 //
/*****************************************************************/

// Inclusion des fichiers header des bibliothèques de fonctions
#include <arduino.h>       // Bibliotheque de fonctions arduino
#include "NEC.h"           // Bibliotheque de fonctions NEC

// Définition des constantes du programme
#define POTAR_LINEAIRE_Pin            A4      // n° de broche du potentiomètre linéaire
#define POTAR_CIRCULAIRE_Pin          A5      // n° de broche du potentiomètre circulaire
#define BOUTON_KLAXON_Pin              8      // n° de broche du bouton poussoir du klaxon
#define LED_INFRAROUGE_Pin             3      // n° de borche de la LED infrarouge
#define AdresseNEC                  0x41      // Adresse du protocole NEC

/*****************************************************************************************************************/
//   La fonction AcquerirPotarVitesse récupère la valeur du potentiomètre linéaire qui définit la vitesse du kart.   //
//                  Elle retourne sa valeur qui est un entier compris entre 0 et 1023.                               //
/*****************************************************************************************************************/

int AcquerirPotarVitesse(void) {

  int valeurPotarLineaire = analogRead(POTAR_LINEAIRE_Pin);
  return valeurPotarLineaire;
}

/*****************************************************************************************************************/
//   La fonction AcquerirPotarRotation récupère la valeur du potentiomètre circulaire qui définit la direction du kart.    //
//                         Elle retourne sa valeur qui est un entier compris entre 0 et 1023.                          //
/*****************************************************************************************************************/

int AcquerirPotarRotation(void) {

  int valeurPotarCirculaire = analogRead(POTAR_CIRCULAIRE_Pin);
  return valeurPotarCirculaire;
}

/*****************************************************************************************************************/
//   La fonction AcquerirBoutonKlaxon récupère l'état du bouton correspondant à la commande du klaxon    //
//    Elle retourne un booléen de valeur True si le bouton est pressé ou False s'il n'est pas pressé     //
/*****************************************************************************************************************/
```

```
bool AcquerirBoutonKlaxon(void) {

  if (digitalRead(BOUTON_KLAXON_Pin) == LOW) {
    return true;
  }

  else {
    return false;
  }
}

/*****************************************************************************************************************************************************************************/
//   La fonction CalculerAdresse calcule l'octet d'adresse, il fusionne les 2 octets de 1 et 7 bits correspondant chacun au klaxon et au numéro d'équipe       //
//              Elle contient 2 paramètres klaxon de type booléen et et numeroEquipe de type byte ainsi qu'une variable adresse de type byte                  //
//               elle retourne adresse, étant un octet de 8 bits, le premier étant l'état du klaxon et les 7 suivants le numéro de l'équipe                   //
/*****************************************************************************************************************************************************************************/

byte CalculerAdresse(bool klaxon, byte numeroEquipe) {

  byte adresse;

  adresse = klaxon << 7 | numeroEquipe;

  return adresse;
}

/*****************************************************************************************************************************************************************************/
//   La fonction CalculerDonnee calcule l'octet de données, il fusionne les 2 octets de 4 bits correspondant chacun à la vitesse et à la direction du kart.    //
//                       Elle contient 2 variables vitesse et rotation subissant une transformation de 1024 valeurs possibles à                             //
//                          seulement 16 et elle retourne un octet de 8 bits contenant d'abord la vitesse puis la direction                                  //
/*****************************************************************************************************************************************************************************/

byte CalculerDonnee(int vitesse, int rotation) {

  byte donnee;

  vitesse = map(vitesse, 0, 1024, 0, 16);
  rotation = map(rotation, 0, 1024, 0, 15);
  donnee = vitesse << 4 | rotation;

  return donnee;
}
```

```
void setup()
{
  // Initialisation du sens de transfert de l'information des broches de type GPIO
  pinMode(POTAR_LINEAIRE_Pin,  INPUT);
  pinMode(POTAR_CIRCULAIRE_Pin, INPUT);
  pinMode(BOUTON_KLAXON_Pin,  INPUT_PULLUP); // Logique inversée
  pinMode(LED_INFRAROUGE_Pin,  OUTPUT);

}

/*****************************************************************************************************************************************************************************/
//   La fonction loop() exécutue en boucle le code se trouvant à l'intérieur                                                                                               //
//                                                                                                                                                                         //
//   Elle initialise 5 variables :                                                                                                                                         //
//                                                                                                                                                                         //
//    - potarVitesse de type int qui stocke la valeur du potentiomètre destiné à la commande de la vitesse donnée par la fonction AcquerirPotarVitesse()                   //
//                                                                                                                                                                         //
//    - potarRotation de type int qui stocke la valeur du potentiomètre destiné à la commande de la direction donnée par la fonction AcquerirPotarRotation()               //
//                                                                                                                                                                         //
//    - klaxon de type booléen qui stocke l'état du klaxon destiné à piloter le klaxon du kart donné par la fonction AcquerirBoutonKlaxon()                                //
//                                                                                                                                                                         //
//    - adresse de type byte qui stocke l'octet de d'adresse donné par la fonction CalculerAdresse() destiné à être envoyé dans la trame NEC afin de transmettre les informations relatives //
//      au klaxon et au numéro d'équipe au récepteur du kart                                                                                                               //
//                                                                                                                                                                         //
//    - donnee de type byte qui stocke l'octet de donnée donné par la fonction CalculerDonnee() destiné à être envoyé dans la trame NEC afin de transmettre les informations relatives //
//      à la direction et à la vitesse au récepteur du kart                                                                                                                //
//                                                                                                                                                                         //
//   La boucle se termine par l'envoi de la trame NEC générée par la fonction GenererTrameNEC() qui combine l'octet d'adresse et l'octet de donnée afin de l'envoyer au récepteur.  //
//                                                                                                                                                                         //
/*****************************************************************************************************************************************************************************/
```

```
void loop() {

  int potarVitesse;
  int potarRotation;
  bool klaxon;

  byte donnee;
  byte adresse;
  uint32_t t;

  static byte donnee_precedente;
  static byte adresse_precedente = 0;
  static uint32_t t_precedent;

  potarVitesse = AcquerirPotarVitesse();
  potarRotation = AcquerirPotarRotation();
  klaxon = AcquerirBoutonKlaxon();

  donnee = CalculerDonnee(potarVitesse, potarRotation);
  adresse = CalculerAdresse(klaxon, AdresseNEC);
  t = millis();

  if ((adresse != adresse_precedente) || (donnee != donnee_precedente) || (t > (t_precedent + 333))) {

    adresse_precedente = adresse;
    donnee_precedente = donnee;
    t_precedent = t;

    GenererTrameNEC(LED_INFRAROUGE_Pin, adresse, donnee);
  }
}
```

3.3.2 Récepteur

Référence du paragraphe : CDT_RCPT_CAPTEUR

Rédacteur : Clément CACHO Mathis BROUSSE

Relecteur : Mathéo GRILLET

Exigences client vérifiées : EXIG_RCPT_CAPTEUR

Compétences GEII : C1-21,22,23,24,25,26

La fonction **AcquerirTrameNEC** (voir figure 60) permet la vérification de la trame NEC reçue. Cette fonction est fournie dans la bibliothèque NEC fournie dans les documents. Elle prend en entrée le numéro de la broche du récepteur et les octets de la trame correspondant respectivement à l'adresse et à la donnée. Elle renvoie une valeur d'erreur : 0, -1, ou -2. La valeur 0 signifie que la trame NEC est correcte et utilisable, la valeur -1 signifie que la trame contient une erreur, la valeur -2 signifie qu'aucune trame n'a été reçue. La fonction est construite de manière à récupérer les information contenues dans l'octet Adresse et l'octet Donnee par le biais d'un pointeur.

```
int8_t AcquerirTrameNEC(int Broche, uint8_t* ptr_Adresse, uint8_t* ptr_Donnee) // AcquerirTrameNEC : fonction d'acquisition
{
  int8_t entete;
  int16_t adresse;
  int16_t adresse_barre;
  int16_t donnee;
  int16_t donnee_barre;
  int32_t temps;
  *ptr_Adresse = 0x00;
  *ptr_Donnee = 0x00;
  entete = AcquerirEnteteNEC(Broche);
  if (entete < 0)                                    // Controle de timing
    return entete;                                   // Erreur de timing
  adresse = AcquerirOctetNEC(Broche);
  if (adresse < 0)                                   // Controle de timing
    return adresse;                                  // Erreur de timing
  adresse_barre = AcquerirOctetNEC(Broche);
  if (adresse_barre < 0)                             // Controle de timing
    return adresse_barre;                            // Erreur de timing
  donnee = AcquerirOctetNEC(Broche);
  if (donnee < 0)                                    // Controle de timing
    return donnee;                                   // Erreur de timing
  donnee_barre = AcquerirOctetNEC(Broche);
  if (donnee_barre < 0)                              // Controle de timing
    return donnee_barre;                             // Erreur de timing
  if ((adresse + adresse_barre) != 0xFF)             // Controle de transmission
    return -1;                                       // Erreur de transmission
  if ((donnee + donnee_barre) != 0xFF)               // Controle de transmission
    return -1;                                       // Erreur de transmission
  if ((temps = AcquerirFrontDescendantNEC(Broche)) < 0) // Controle de timing
    return temps;                                    // Erreur de timing
  *ptr_Adresse = adresse;
  *ptr_Donnee = donnee;
  return 0;                                          // Trame correcte
}
```

Figure 60 : Programme de la fonction AcquerirTrameNEC

Référence du paragraphe : CDT_RCPT_TRAITEMENT

Rédacteur : Clément CACHO Mathis BROUSSE

Relecteur : Mathéo GRILLET

Exigences client vérifiées : EXIG_RCPT_TRAITEMENT , EXIG_RCPT_SECURITE

Compétences GEII : C1-21,22,23,24,25,26

Nous avons dans un premier temps inclus les bibliothèques nécessaires à l'utilisation de certaines fonctions. (voir figure 61)

```
8  // inclusion des fichiers header des bibliothèques de fonctions
   Arduino
9  #include <stdint.h>
10 #include <arduino.h>
11 #include "NEC.h"
```

Figure 61 : Inclusion des bibliothèques

Nous avons par la suite défini les noms de nos broches en fonction du nombre de la broche associée puis nous avons défini notre numéro d'équipe. (voir figure 62)

```
14  // definition des constantes du projet
15  #define Servomoteur_Pin              11
16  #define Moteur_Pin                   10
17  #define Buzzer_Pin                    9
18  #define LedBleue_Pin                 A1
19  #define Recepteur_IR_Pin              3
20  #define NumeroEquipe               0x41
```

Figure 62 : Définition des variables et broches

Nous avons programmé plusieurs programmes liés au traitement de l'information.

Dans un premier temps nous avons programmé la fonction **ExtraireNumeroEquipe**(uint8_t adresse). La fonction prend en entrée l'octet d'adresse de la trame NEC et retourne en sortie une valeur du numéro d'équipe d'où provient la trame. Les bits d'adresse de l'équipe sont les 7 bits après le bit de poids fort. Nous pouvons donc le récupérer à l'aide d'un opérateur binaire "et" (&). (Voir figure 63)

```
31  uint8_t ExtraireNumeroEquipe(uint8_t Adresse) { // retourne une
    valeur : [41, 43 ]
32    Num_Equipe = Adresse & 0b01111111;
33    return Num_Equipe ;
34  }
```

Figure 63 : Programmation de la fonction ExtraireNumeroEquipe

Nous avons ensuite programmé la fonction **ExtraireAngle**(uint8_t Donnee). La fonction prend en entrée l'octet de donnée de la trame NEC et renvoie une valeur correspondante à la direction souhaitée des roues. La direction des roues est donnée sur les 4 bits de poids faible, nous pouvons donc les récupérer de la même manière que précédemment, par l'utilisation d'un opérateur &. (Voir figure X) Nous récupérons alors une valeur comprise entre 0 et 14 afin d'obtenir un nombre impair de valeur possible nous permettant de définir une valeur de milieu. Finalement nous allons la convertir en une valeur comprise entre -100 et 100 pour une lecture simplifiée du programme. Pour ce faire nous pourrons utiliser la fonction map() en remplissant les paramètres selon les valeurs souhaitées. (Voir figure 64)

```
37 uint8_t ExtraireAngle(uint8_t Donnee) { // retourne une valeur : [
   -100 ; 100 ]
38   position = Donnee & 0b00001111 ;
39   position = map(Position, 0, 14, -100, 100)
40              return position ;
41 }
```

Figure 64 : Programmation de la fonction ExtraireAngle

Par la suite nous avons programmé la fonction **ExtraireVitesse**(uint8_t Donnee). La fonction prend en entrée l'octet de donnée et renvoie une valeur correspondante à la puissance donnée au moteur. L'information de la puissance se situe sur les 4 bits de poids fort, nous pouvons les récupérer par l'opérateur binaire >>. Finalement nous pouvons convertir la valeur donnée par la trame NEC en une valeur comprise entre 0 et 100 par l'utilisation de la fonction map(). (Voir figure 65)

```
44 uint8_t ExtraireVitesse(uint8_t Donnee) { // retourne une valeur : [
   0 ; 100 ]
45   pourcentage = Donnee >> 4 ;
46   pourcentage = map(pourcentage, 0, 15, 0, 100)
47                 return pourcentage ;
48 }
```

Figure 65 : Programmation de la fonction ExtraireVitesse

Nous avons ensuite programmé la fonction **ExtraireEtatBuzzer**(uint8_t Adresse). La fonction **ExtraireEtatBuzzer** permet la lecture de l'information liée au buzzer. Elle prend en entrée l'octet d'adresse et retourne la valeur donnée au fonctionnement du buzzer : 0 si inactif ou 1 si actif. Le bit lié au buzzer étant donné en bit de poid fort sur l'octet de l'adresse on peut utiliser l'opérateur logique ">>" pour décaler les bits et ainsi récupérer le bit de poid fort. (Voir figure 66)

```
51  uint8_t ExtraireEtatBuzzer(uint8_t Adresse) { // retourne : 0
    (inactif), 1 (actif)
52    etat = Adresse >> 7 ;
53    return etat;
54  }
```

Figure 66 : Programmation de la fonction ExtraireEtatBuzzer

Par la suite nous avons programmé la fonction **setup**(). La fonction **setup**() nous permet de définir les broches utilisées en tant que sortie ou entrée. La fonction **setup**() ne prend rien en entrée et ne renvoie rien en sortie. Les broches de commande du servomoteur, du moteur, du buzzer, de la LED bleue sont des sorties et la broche du récepteur infrarouge est une entrée. Nous avons donc défini les broches de la manière suivante :

```
98  void setup(void) {
99    pinMode(Servomoteur_Pin, OUTPUT);
100   pinMode(Moteur_Pin, OUTPUT);
101   pinMode(Buzzer_Pin, OUTPUT);
102   pinMode(LedBleue_Pin, OUTPUT);
103   pinMode(Recepteur_IR_Pin, INPUT);
104   myservo.attach(9);
105   moteur.attach(11);
106   moteur.write(0);
107   delay(7000);
108 }
```

Figure 67 : Programmation de la fonction setup()

(Les lignes 104 à 107 sont détaillées dans une partie suivante)

La fonction **loop**() (voir figure 68) permet le traitement de l'information liée à la trame NEC afin de commander les actions (voir figure 28). Elle ne prend rien en entrée et ne renvoie rien. Premièrement on définit les variables locales : uint8_t Donnee; uint8_t Adresse; uint8_t pourcentage; uint8_t position; uint8_t etat; uint8_t erreur; uint8_t Num_Equipe (de type unsigned int sur 8 bits). Premièrement nous vérifions la trame NEC à l'aide de la fonction **AcquerirTrameNEC** en attribuant la valeur retournée à la variable erreur.

Par l'utilisation de la fonction **if** (condition) nous ouvrons une partie du programme dans le cas où erreur vaut 0. Dans ce cas, nous lisons le numéro de l'équipe en affectant la valeur retournée par la fonction **ExtraireNumeroEquipe**. De la même manière, si le numéro d'équipe correspond à notre équipe (en l'occurrence 41) nous réalisons les actions. Nous

allumons la LED Bleue (témoin de réception d'une trame NEC) par la fonction **PiloterLedATransmissionNEC**. Nous affectons à la variable etat l'état lu par la fonction **ExtraireEtatBuzzer** puis nous actionnons le buzzer par la fonction **PiloterBuzzer** en fonction de l'état etat. Nous affectons à la variable pourcentage le pourcentage de puissance lu par la fonction **ExtraireVitesse** puis nous actionnons le moteur à l'aide de la fonction **PiloterMoteur**. Finalement nous affectons à la variable position la valeur de l'angle des roues lue par la fonction **ExtraireAngle** puis nous actionnons la rotation des roues par la fonction **PiloterRoues**.

Dans le cas où le numéro d'équipe ne correspond pas au nôtre, nous ouvrons une partie du programme afin d'éteindre la LED Bleue et le moteur par les fonctions **PiloterLedATransmissionNEC** et **PiloterMoteur**.

De la même manière si l'erreur de la trame NEC n'est pas égale à 0 nous ouvrons une partie du programme afin d'éteindre la LED Bleue et le moteur par les fonctions **PiloterLedATransmissionNEC** et **PiloterMoteur**. (voir figure 68)

```
void loop(void) {
  uint8_t Donnee;
  uint8_t Adresse;
  uint8_t pourcentage;
  uint8_t position;
  uint8_t etat;
  uint8_t erreur;
  uint8_t Num_Equipe;
  erreur = AcquerirTrameNEC(&Adresse, &Donnee);
  if (erreur == 0) {
    Num_Equipe = ExtraireNumeroEquipe(Adresse);
    if (Num_Equipe == 0x41) {
      PiloterLedATransmissionNEC(1);
      etat = ExtraireEtatBuzzer(Adresse);
      PiloterBuzzer(etat);
      pourcentage = ExtraireVitesse(Donnee);
      PiloterMoteur(pourcentage);
      position = ExtraireAngle(Donnee);
      PiloterRoues(position);
    }
    else {
      PiloterLedATransmissionNEC(0);
      PiloterMoteur(0);
    }
  }
  else {
    PiloterLedATransmissionNEC(0);
    PiloterMoteur(0);
  }
}
```

Figure 68 : Programmation de la fonction loop

Référence du paragraphe : CDT_RCPT_RETENTISSEMENT

Rédacteur : Clément CACHO Mathis BROUSSE

Relecteur : Mathéo GRILLET

Exigences client vérifiées : EXIG_RCPT_RETENTISSEMENT

Compétences GEII : C1-21,22,23,24,25,26

Référence du paragraphe : CDT_RCPT_MOTEUR

Rédacteur : Mathéo GRILLET

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées : EXIG_RCPT_MOTEUR

Compétences GEII : C1-21,22,23,24,25,26

Conformément au diagramme d'architecture informatique du récepteur (figure 27 présent dans le paragraphe CPR_RCPT_ARCHI_INFO) nous définissons une fonction "PiloterMoteur" cette fonction nécessite un pourcentage en paramètre.

La commande du moteur est définie de la même manière qu'un servo moteur. C'est pourquoi nous utiliserons la librairie Servo.h. Nous définissons en début de code, dans la partie setup, "Servo moteur". Cette ligne de code permet d'associer "moteur" à l'utilisation de la librairie. On peut donc écrire "moteur.action_de_la_librairie_servo(paramètre de la fonction associé)".

Par lecture de la description de la fonction, pour définir un pin on utilisera "`attach(pin, min, max)`" . les paramètres sont définis de la manière suivante:

paramètre	description
pin	le numéro de pin auquelle le moteur est connecté (le n° de pin n'est pas celui de l'ATMEGA328P mais celui associé sur la carte arduino)
min	(Optionnel) la largeur d'impulsion, en microsecondes, correspondant à l'angle minimum (0 degré) sur le servo (par défaut : 544)
max	(Optionnel) la largeur d'impulsion, en microsecondes, correspondant à l'angle maximum (180 degrés) sur le servo (2400 par défaut)

Finalement, on ne modifiera pas les paramètres et on utilisera les paramètres par défaut. On écrira donc: `moteur.attach(11);`

Pour ne pas reconfigurer le moteur, nous devons débuter à une vitesse de 0. C'est la fonction moteur.write(angle) qui permet de la faire. Le paramètre angle est une valeur comprise entre 0 et 180°. 0 étant la vitesse minimum et 180 la vitesse maximum. Etant donné que nous souhaitons que

cela soit réalisé uniquement lors de la mise sous tension du récepteur nous mettrons donc moteur.write(0) et un délai de 7 secondes delay(7000) dans la partie setup.

Résumé de la partie setup nécessaire à l'utilisation de cette fonction figure 69.

```
#include <Servo.h>
Servo moteur;
void setup() {
  moteur.attach(11);
  moteur.write(0);
  delay(7000);
}
```

Figure 69 : Setup pour piloter le moteur

Pour commander la vitesse de rotation du moteur nous utilisons la fonction moteur.write(angle). Nous devons donc convertir le pourcentage en une valeur comprise entre 0 et 180°. On utilisera donc la fonction map().

Résumé de la fonction PiloterMoteur en figure 70.

```
void PiloterMoteur(int pourcentage) {
  moteur.write(map(pourcentage, 0, 100, 0, 180));
}
```

Figure 70 : fonction piloter moteur

Conformément au diagramme d'architecture informatique du récepteur nous avons bien une fonction PiloterMoteur() avec comme paramètre un pourcentage.

Référence du paragraphe : CDT_RCPT_ROUE

Rédacteur : Mathéo GRILLET

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées : EXIG_RCPT_ROUE

Compétences GEII : C1-21,22,23,24,25,26

Lors de la conception préliminaire nous avons convenu d'une fonction PiloterRoues() avec pour paramètre une position comprise entre -100 et 100. - 100 étant une rotation de 30 ° à gauche, 0 les roues en position "tout droit" enfin 100 une rotation de 30° à droite.

Les roues sont pilotées par un servo moteur nous utiliserons la même librairie que pour le moteur c'est-à-dire servo.h.

Les paramètres d'initialisation sont les mêmes que pour le moteur. La fonction setup est donc définie comme sur la figure 71.

```
5   Servo myservo;
6   void setup() {
7     myservo.attach(9);
```

Figure 71: Setup pour piloter les roues

Pour piloter le servo moteur nous utiliserons la fonction myservo.write(angle). Nous souhaitons une rotation des roues de 30° de chaque côté. Nous avons effectué des tests:

- 0° sur le servo moteur correspond à une rotation de 30° à gauche des roues.
- 90 ° sur le servo moteur correspond aux roues droites.
- 180° sur le servo moteur correspond à une rotation de 30° à droite des roues.

Nous commandons donc de 0° à 180° donc pour avoir un angle de -30° à 30° de rotation des roues. Pour cela nous n'utilisons pas une fonction map mais nous allons réaliser une fonction mathématique que nous avons définie. Il s'agit d'une fonction carré afin d'avoir plus de valeur associé proche de 0° (tout droit) permettant ainsi un pilotage plus habile.

Avec cette méthode l'effet que cela aura sur l'expérience utilisateur est que la commande de direction sera moins sensible aux alentours du "tout droit".

Pour un souci de performance, on ne réalise pas les calculs entre -100 et 100. Cela ne posera pas de problème en termes de précision puisque nous recevons au départ (à l'acquisition) une valeur entre 0 et 14. Nous pouvons donc repasser dans cette plage de valeurs. On utilisera map. Une fois notre nouvelle plage de valeur définie 0; 14 on peut réaliser l'association de chacune d'entre elles à un angle.

La fonction retenue sera donc la suivante: 90-(A2-7)^2*1,83673469 avant 7 et =(A10-7)^2 * 1,83673469 + 90 après 7. Le graphique d'association des angles sera donc celui présent sur la figure 72.

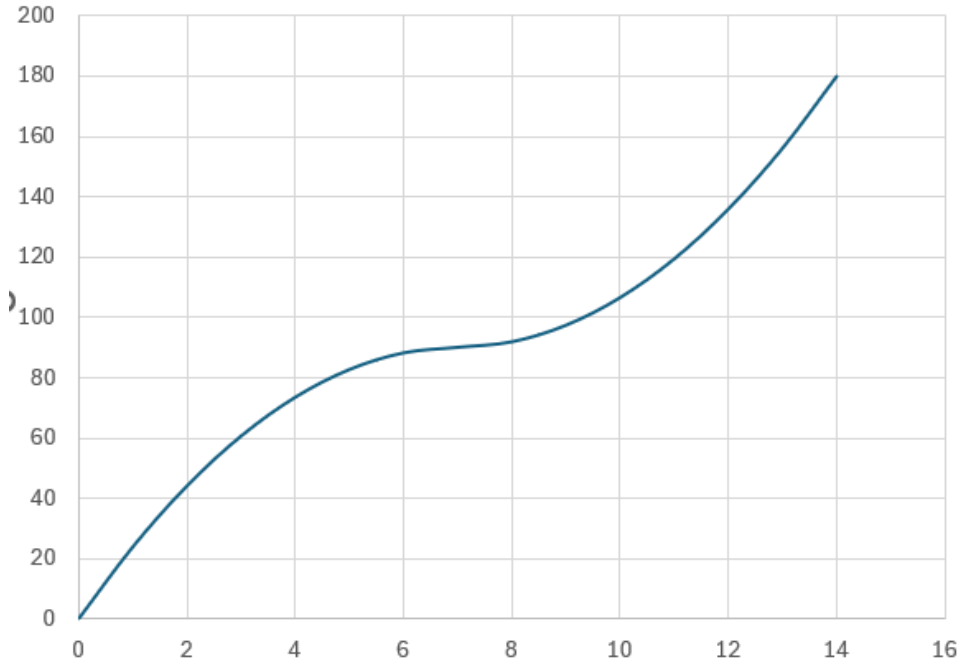


Figure 72: Graphique d'association de l'angle par rapport aux paramètres de la fonction renseigné

La fonction sera définie de la manière suivante (voir figure 73):

```
38  void PiloterRoues(float positionini) {
39    float position = map(positionini, -100, 100, 0,14);
40    int position_formatsortie;
41    if (position >= 7) {
42      position_formatsortie = (position - 7) * (position - 7) * 1.83673469 + 90;
43    } else {
44      position_formatsortie = 90 - (position - 7) * (position - 7) * 1.83673469 ;
45    }
46    myservo.write(position_formatsortie);
47  }
```

Figure 73: fonction piloter roues

En conclusion on a bien une fonction qui associe un angle de -30° à 30° aux roues à partir d'une valeur comprise entre -100 et 100.

Référence du paragraphe : CDT_RCPT_CONNEXION

Rédacteur : Mathéo GRILLET

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées : EXIG_RCPT_CONNEXION

Compétences GEII : C1-21,22,23,24,25,26

Dans cette fonction, on souhaite allumer la LED lorsque le paramètre est égale à 1. Pour cela on réalisera un simple test du paramètre et on pilotera les GPIO sur lequel est connecté la LED en logique positive. On utilisera la fonction digitalWrite(). Voir le code de pilotage de la LED figure 74.

```
49  void PiloterLedTransmitissionNEC(uint8_t etat) {
50    if (etat == 1) {  // paramètre etat = 1
51      digitalWrite(A1, HIGH);
52    } else  // paramètre etat = 0
53    {
54      digitalWrite(A1, LOW);
55    }
56  }
```

Figure 74 : Code de pilotage de la LED

Il est important de définir la broche en tant que broche de sortie dans le setup (voir figure 75).

```
10    pinMode(A1, OUTPUT);
```

Figure 75 : Définition de la broche en sortie

Référence du paragraphe : CDT_RCPT_KLAXON

Rédacteur : Mathéo GRILLET

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées : EXIG_RCPT_KLAXON

Compétences GEII : C1-21,22,23,24,25,26

L'objectif de cette partie est de réaliser une fonction permettant d'émettre un son avec le buzzer lorsque le paramètre est 1. On utilisera donc la fonction tone permettant d'émettre un son avec une certaine fréquence ici 4000 Hz. Le code sera donc le suivant (voir Figure 76).

```
28  void PiloterBuzzer(int etat){
29    if(etat == 1){
30      tone(9, 4000, 1000);
31    }else{
32      digitalWrite(9, LOW);
33    }
```

Figure 76 : Code du buzzer

On définit également la broche en tant que sortie dans la partie setup (voir figure 77).

```
8    pinMode(9, OUTPUT);
```

Figure 77 : Code du buzzer partie setup

Référence du paragraphe : CDT_RCPT_LOGO

Rédacteur : TISSOT Nicolas

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées : EXIG_RCPT_LOGO

Compétences GEII : C1-21,22,23,24,25,26

Figure 78: Logo

Pour la création de ce logo, nous avons utilisé le site “bing IA”.

Les génération d’image par l’IA ne sont pas encadrées par la loi sur le droit d’auteur, nous sommes donc en accord avec la législation. Nous pouvons donc utiliser cette image en tant que logo.

Référence du paragraphe : CDT_RCPT_DELAI

Rédacteur : TISSOT Nicolas

Relecteur : Clément CACHO Mathis BROUSSE

Exigences client vérifiées : EXIG_RCPT_DELAI

Compétences GEII : C1-21,22,23,24,25,26

Suite à la conception détaillée du projet et vérification de la conformité des exigences, nous pouvons affirmer que le planning de conception est respecté.

Référence du paragraphe : CPR_COUT

Rédacteur : DELANNOY Ylhan

Relecteur : CACHO Clément et BROUSSE Mathis GRILLET Mathéo TISSOT Nicolas, GIBELIN Thomas, Maxence CORDEAU,

Exigences client vérifiées par pré-conception : EXIG_COUT

Compétences GEII : C1-21,22,23,24,25,26

Voici le coût de notre projet après la conception détaillé :

Coût du projet

Pour conclure sur le coût du projet nous rentrons bien dans le budget autorisé pour réaliser le KAH qui est de 120 euro sachant que nous avons un coût de 99 euros TTC.

4. Schémas électriques de la conception détaillée du produit

Rédacteur : TISSOT Nicolas/GIBELIN Thomas Delannoy Ylhan Cordeau Maxence

Relecteur :

La conception détaillée nous a permis d'élaborer un schéma électrique pour la partie émetteur ainsi que la partie récepteur. Nous avons, à partir des exigences, dimensionné chaque composants utilisés avec une justification détaillée.

Les dérisquage prouvent la conformité des dimensionnement réalisé.

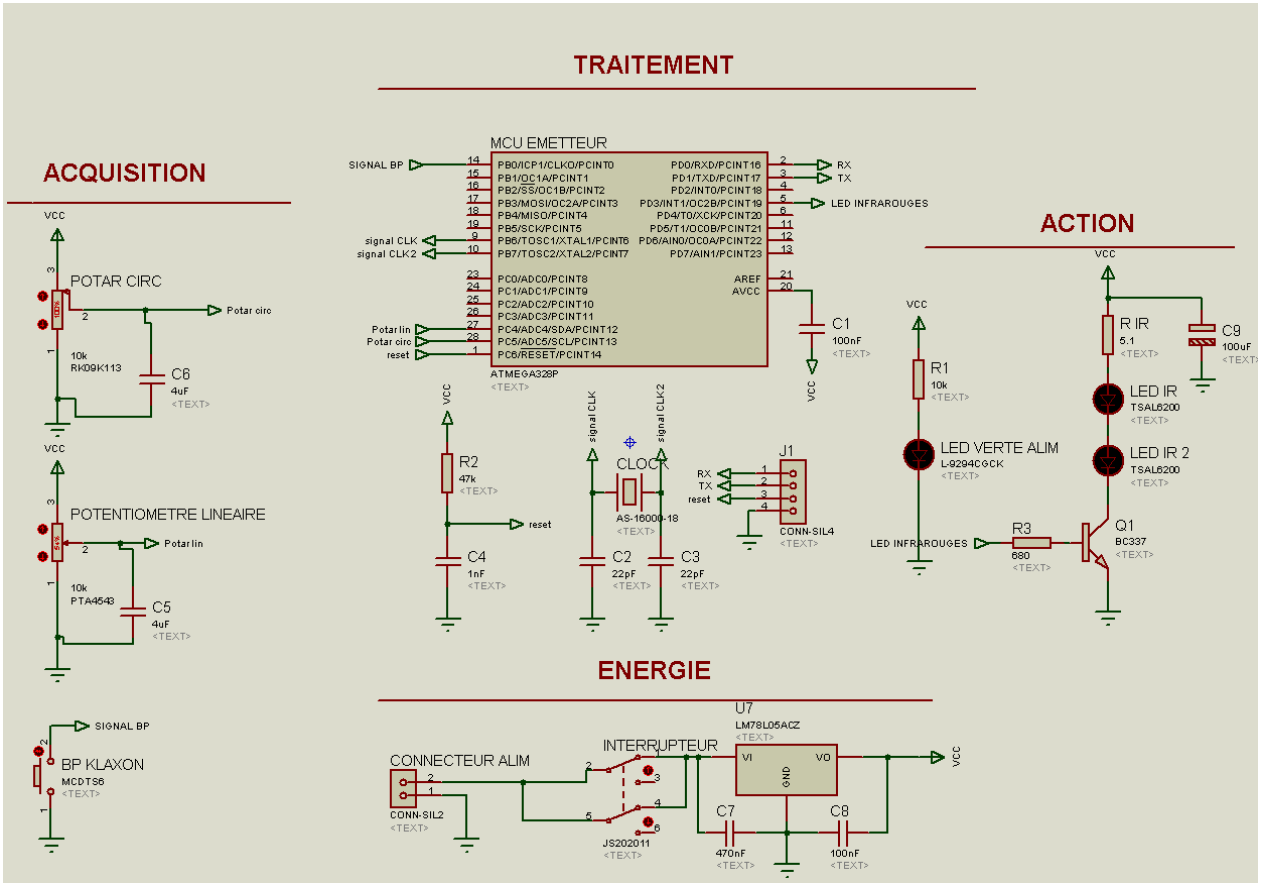


Figure 79 : schéma électrique final de l'émetteur

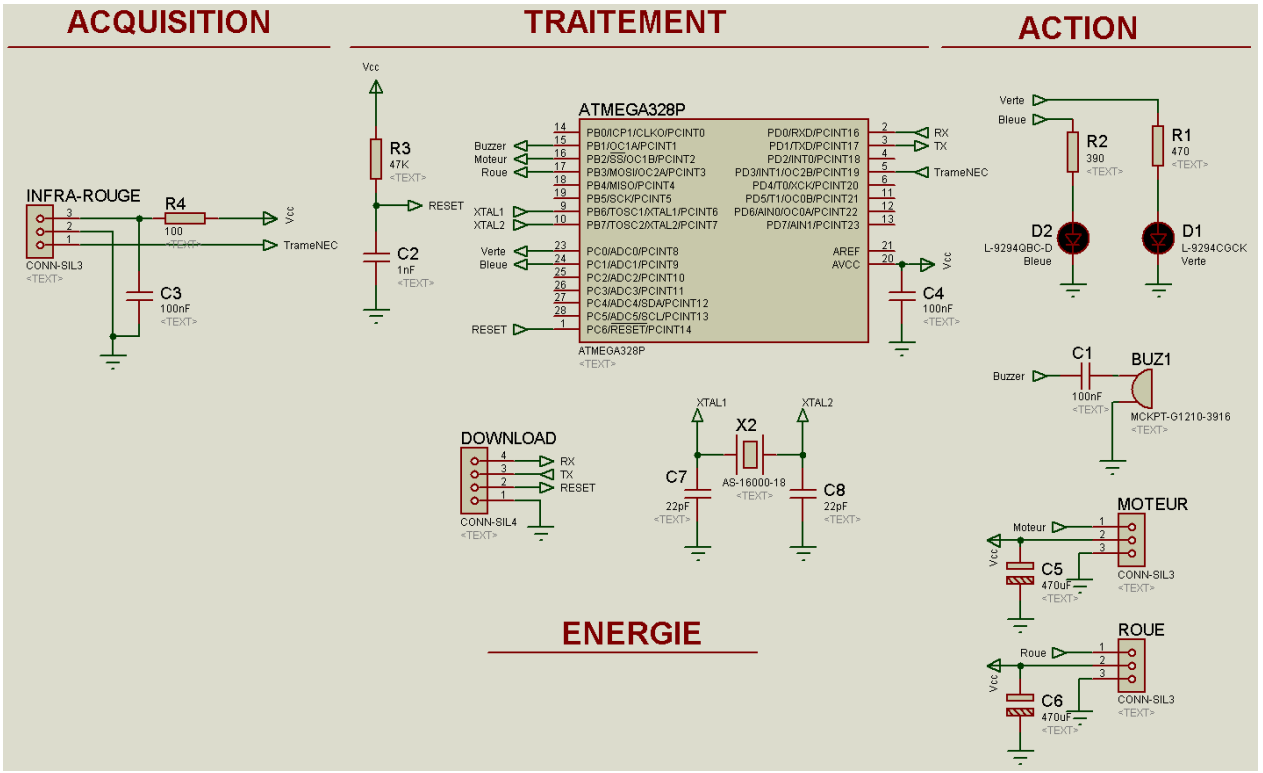


Figure 80 : schéma électrique final du récepteur

5. Conclusion de la conception du produit

Rédacteur : TISSOT Nicolas/GIBELIN Thomas

Relecteur : Delannoy Ylhan Cordeau Maxence

Suite à la conception préliminaire puis détaillée du produit, nous pouvons affirmer que chaque exigence est respectée.

Nous pouvons donc commencer la phase de fabrication du prototype.

6. 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 de développement	Paragraphes en lien avec l'exigence	Statut
EXIG_EMTT_DIMENSIONS	Conception préliminaire	**CPR_EMTT_DIMENSIONS**	Conforme
EXIG_EMTT_LOGO	Conception préliminaire	**CPR_EMTT_LOGO**	Conforme
EXIG_EMTT_ENERGIE	Conception préliminaire Conception détaillée	**CPR_EMTT_ENERGIE** **CDT_EMTT_ENERGIE**	Conforme Conforme
EXIG_EMTT_INTERRUPTEUR	Conception préliminaire	**CPR_EMTT_INTERRUPTEUR**	Conforme
EXIG_EMTT_IHM	Conception préliminaire Conception détaillée	**CPR_EMTT_IHM** **CDT_EMTT_IHM**	Conforme Conforme
EXIG_EMTT_KLAXON	Conception préliminaire Conception détaillée	**CPR_EMTT_KLAXON** **CDT_EMTT_KLAXON**	Conforme Conforme
EXIG_EMTT_TRAITEMENT	Conception préliminaire Conception détaillée	**CPR_EMTT_TRAITEMENT** **CDT_EMTT_TRAITEMENT**	Conforme Conforme
EXIG_EMTT_REPETITIVITE	Conception préliminaire Conception détaillée	**CPR_EMTT_REPETITIVITE**	Conforme Conforme
EXIG_EMTT_RETENTISSEMENT	Conception préliminaire	**CPR_EMTT_RETENTISSEMENT**	Conforme
EXIG_EMTT_PUISSANCE	Conception préliminaire Conception détaillée	**CPR_EMTT_PUISSANCE** **CDT_EMTT_PUISSANCE**	Conforme Conforme
EXIG_EMTT_INDICATEUR	Conception préliminaire Conception détaillée	**CPR_EMTT_INDICATEUR** **CDT_EMTT_INDICATEUR**	Conforme Conforme

Exigence	Méthodes de développement	Paragraphes en lien avec l'exigence	Statut
EXIG_RCPT_DIMENSIONS	Conception préliminaire Conception détaillée	**CPR_RCPT_DIMENSIONS**	Conforme
EXIG_RCPT_LOGO	Conception préliminaire Conception détaillée	**CPR_RCPT_LOGO**	Conforme
EXIG_RCPT_ENERGIE	Conception préliminaire Conception détaillée	**CPR_RCPT_ENERGIE** **CDT_RCPT_ENERGIE**	Conforme Conforme
EXIG_RCPT_INTERRUPTEUR	Conception préliminaire	**CPR_RCPT_INTERRUPTEUR**	Conforme
EXIG_RCPT_CAPTEUR	Conception préliminaire Conception détaillée	**CPR_RCPT_CAPTEUR** **CDT_RCPT_CAPTEUR**	Conforme Conforme
EXIG_RCPT_TRAITEMENT	Conception préliminaire Conception détaillée	**CPR_RCPT_TRAITEMENT** **CDT_RCPT_TRAITEMENT**	Conforme Conforme
EXIG_RCPT_SECURITE	Conception préliminaire	**CPR_RCPT_SECURITE**	Conforme
EXIG_RCPT_RETENTISSEMENT	Conception préliminaire	**CPR_RCPT_RETENTISSEMENT**	Conforme
EXIG_RCPT_MOTEUR	Conception préliminaire Conception détaillée	**CPR_RCPT_MOTEUR** **CDT_RCPT_MOTEUR**	Conforme Conforme
EXIG_RCPT_ROUE	Conception préliminaire Conception détaillée	**CPR_RCPT_ROUE** **CDT_RCPT_ROUE**	Conforme Conforme
EXIG_RCPT_INDICATEUR	Conception préliminaire Conception détaillée	**CPR_RCPT_INDICATEUR** **CDT_RCPT_INDICATEUR**	Conforme Conforme
EXIG_RCPT_CONNEXION	Conception préliminaire Conception détaillée	**CPR_RCPT_CONNEXION** **CDT_RCPT_CONNEXION**	Conforme Conforme

Exigence	Méthodes de développement	Paragraphes en lien avec l'exigence	Statut
EXIG_RCPT_KLAXON	Conception préliminaire Conception détaillée	**CPR_RCPT_KLAXON** **CDT_RCPT_KLAXON**	Conforme Conforme
EXIG_DELAI	Conception préliminaire Conception détaillée	**CPR_DELAI** CDT_RCPT_DELAI	Conforme
EXIG_COUT	Conception préliminaire Conception détaillée	**CPR_COUT**	Conforme