POLYTECH Montpellier

Stage industriel de fin d'études

Département MEA

ANNEE 2019 - 2020

Mangeoire Connectée

Quentin Lehérissé

STAGE

MEA

ECOLE POLYTECH Montpellier
UNIVERSITE DE MONTPELLIER – Campus Triolet
Place Eugène Bataillon 34095 MONTPELLIER CEDEX 5
Tél. : 04 67 14 31 60 – Fax : 04 67 14 45 1 E-mail : scola@polytech.univ-montp2.fr

POLYTECH MONTPELLIER

Table des matières

0-Avant-propos

Je remercie la SATT AxLR pour son accueil et son partenariat avec Polytech s'agissant de ce stage de fin d'étude très enrichissant tant sur le plan de la conception hardware que software. Mes remerciements vont aussi particulièrement à Monsieur Laurent Latorre qui a su être présent pour me guider dans certaines situations délicates sur le plan technique et notamment dans un contexte de crise sanitaire. Cette crise sanitaire liée au covid-19 aura été une contrainte agissant sur mes possibilités d'actions, il aura donc fallu lors de la période de confinement organiser un planning afin de minimiser l'impact de ce contexte.

1-Présentation de l'entreprise

L'entreprise a été fondée en 2012, basée dans l'université de Montpellier II950 Rue St - Priest, 3409

0 Montpellier. La SATT axlr une société d'accélération du transfert de technologies (SATT). Ils sont spécialisés dans la maturation et la commercialisation de projets innovants issus de la recherche académique. Ils interagissent avec la majeure partie des laboratoires de la recherche publique implantés sur l'arc méditerranéen en Occitanie. Cet région française et européenne est l'une des plus dynamiques, avec plus de 200 laboratoires et près de 12 000 chercheurs. Le président de cette SATT est monsieur Nerin Philippe, ancien Directeur de recherche chez Horiba Medical. Pour lui, l'objectif de la SATT est de : " de permettre aux start-ups d'atteindre une croissance plus forte, en les conseillant notamment dans l'adéquation entre leur produit et le marché. L'accompagnement ira jusqu'à la signature du premier client. De même, nous voulons travailler sur la modélisation financière des deep techs, en les amenant à afficher les bons indicateurs quand elles entrent en phase de levée de fonds".

La SATT en chiffre, c'est :

- 992 projet évalué
- 161 programmes d'innovations lancés
- 47M€ investis
- 79 contrats de transferts vers des entreprises
- 87 start-ups accompagnées
- 233 dossiers de propriétés intellectuelles valorisés

2-Introduction

Je vous présente mon stage en tant qu'ingénieur maturation chez la SATT AxLR, une entreprise qui accompagne justement à la maturation de projet/invention. Ma mission a été de faire parvenir le projet Mangeoire connecté vers une solution complète et fonctionnelle. La Mangeoire connectée est un outil de recensement aviaire. Cette Mangeoire connectée est constituée d'un socle solide qui constitue la Mangeoire là où l'oiseaux ira se nourrir et la partie recensement outillée, la partie qui détecte pèse prend en photo l'oiseaux et stocke les données. C'est un outil qui doit être selon le cahier des charges autonome en énergie, bien sûr autonome dans la partie recensement ainsi la seule partie où l'autonomie n'est pas complète est le stockage des données, les données étant stockés sur une carte SD, il faudra donc penser à collecter les données sur cette carte. Pour permettre à la mangeoire connectée de remplir toute ces fonctionnalités, nous avons besoins de plusieurs modules qui sont : un module caméra afin de prendre les photos, un module SD afin de pourvoir stockée les données, un module de pesée afin de prendre le poids de l'oiseau, un piézoélectrique afin de détecter la pose de l'oiseau sur la mangeoire, un module GPS pour connaître l'heure, la date et l'emplacement géographique enfin un module Bluetooth pour connaitre l'état de remplissage de la carte SD. En ce qui concerne la partie énergie, l'énergie sera collectée par un panneau solaire puis grâce à un module électrique sera stockée dans une batterie. Grâce aux fonctions de sommeil du microcontrôleur stm32L496, et une activation stratégique des dispositifs, il nous est donc possible d'économiser l'énergie des batteries afin de permettre l'autonomie du module car l'énergie produite par un panneau solaire est périodique et limitée, la solution de l'économie d'énergie est préférée à un surdimensionnement des panneaux et des batteries car cela augmenterait drastiquement le coup de production de la mangeoire. Deux équipes ont déjà travaillées sur ce projet, une a réalisé un montage qui permet de stocker l'énergie électrique d'un panneau dans une batterie l'autre à élaborer une carte (et l'informatique qui va avec) qui permet de peser, photographier et stocker avec une fonction aussi de mise en veille. Je dois donc réaliser fusion intelligente des deux solutions, corriger les éventuels problèmes de conception (tel que l'inversion de certaine patte, et résoudre certain problème d'alimentation via des transistors), ajouter sur le hardware deux modules GPS et Bluetooth Low Energy (BLE) et enfin travailler sur le software permettant grâce à un OS temps réel permettre à toutes ses fonctionnalités de travailler ensemble. Il sera aussi présenté certaines étapes intermédiaires comme l'explication des stratégies de fonctionnement ainsi que des choix technologiques et rétrospectivement aussi sera étudié les choix technologiques des équipes précédentes (choix du microcontrôleur …).

3-Rétrospective technique

Première partie : Collecte d'énergie solaire

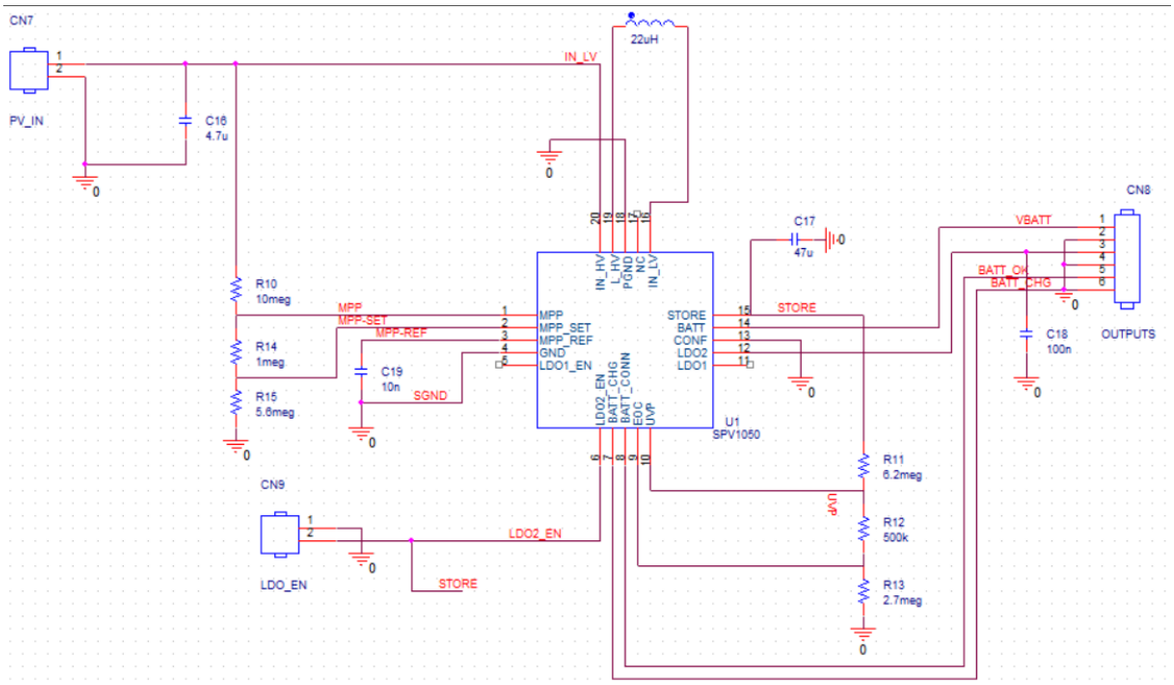


FIGURE 1 SCHEMA COLLECTE D'ENERGIE VENANT DES PANNEAUX PHOTOVOLTAÏQUE

La première équipe dont nous allons résumer le travail est celle qui s'est attelée à convertir le courant électrique des panneaux solaires en un courant capable de recharger une batterie. L'équipe a utilisé le montage représenté par le schéma ci-dessus, le module principal est le SPV1050.

Ce SPV1050 est un montage BUCKBOOST, il sert à réduire ou augmenter la tension de sortie par rapport à l'entrée afin de pouvoir charger alimenter convenablement en courant une batterie, etc… Les résistances montées autour de ce module servent à le calibrer. On choisit les résistances R10, R14 et R15 afin de traquer le MPPT (maximum power point tracking). Ce point sert à suivre à ajuster la tension et le courant qu'il tire afin de retirer le maximum de puissance du panneau solaire.

On peut également noter que le SPV1050 prévoit des LDO (sortie de courant) de tension 1.8 et 3.3 V mais nous ne les utiliserons pas.

La capacité C16 a pour rôle d'être un petit réservoir à charge, c'est pour cela que toute les 16 secondes pendant 400ms le SPV1050 cesse de commuter pour permettre à cette capacité de se recharger.

Les résistances R11, R12 et R13 servent à calibrer la tension de sortie.

Deuxième partie : prise de photo et stockage à déclenchement par impulsion d'un piézoélectrique

La seconde équipe à travailler sur une carte électronique dont le microprocesseur est un stm32l496.

Le choix de ce microcontrôleur est aussi le mien car les stm32LXXX sont spécialisés pour les utilisations low-power, la mise hors tension de la plupart des fonctionnalités de la carte est ainsi possible ainsi qu'il est possible de mettre ce microcontrôleur en mode shutdown, un mode qui le rapproche grandement de la complète mise hors tension en effet tout est mis hors tension sauf le RTC et la pin PC13.

C'est dans cette optique que l'équipe a utilisé la pin PC13 afin de pouvoir réveiller le microcontrôleur grâce à l'impulsion d'un piézoélectrique.

Pour la prise de photo a été utilisé le module OV2640, ce module permet la compression JPEG et à une liaison parallèle, les liaisons série (MPI généralement) ne sont pas supporter sur stm32.

Les protocoles utilisés pour communiquer avec ce module sont le SCCB (proche de l'I2C), et le DCMI.

Je vais expliquer succinctement le fonctionnement de ces deux protocoles, informations tirées d'un état de l'art sur les quelques protocoles caméras, (la description du protocole MIPI sera en annexe).

DCMI

Le digitale caméra interface (ou DCMI) est un bus de data parallèle pour stm32. Le bus parallèle est composé d'au maximum 14 lignes de data (8 ,10 ,12 ,14), l'horloge des pixels DCMI_PIXCLK, DCMI_HSYNC (synchronise horizontalement), DCMI_VSYNC (synchronise verticalement).

Le bloc DCMI est composé de 4 composants principaux :

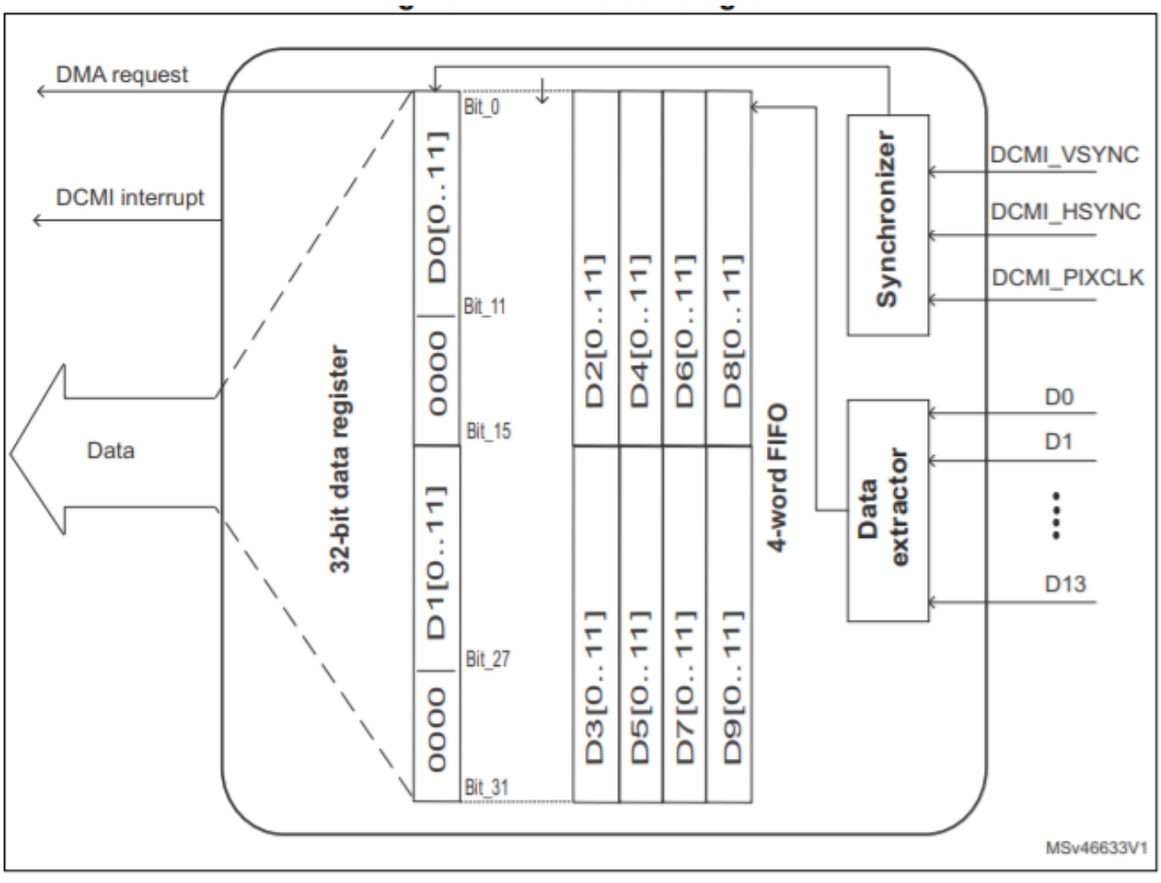


FIGURE 2 DIAGRAMME DE BLOC DU DCMI

Le **synchroniseur DCMI (synchronizer)** ordonne le du flux de donnée.

L'**extracteur de data (data extractor)** reçoit la data venant du bus parallèle.

Le **FIFO (4-word FIFO)** : groupe-la data et sert à adapter le taux de transfert vers le bus AHB.

Le **registre 32bits (32-bit register)** : c'est un registre où la data est placée pour être transférée vers la DMA channel.

Le registre est rempli de manière différente selon le nombre de ligne de data présente sur le bus.

Lorsqu'il y a 8 lignes de data, les 8 premiers bits sont placés à la place des LSB dans le registre puis vient les 8 seconds, jusqu'aux 8 quatrièmes qui sont à la place des MSB.

Lorsqu'il y a X lignes (X=10 ou plus) de data, il n'y a que 2 séries de bits placées, la première de 0 à X et la deuxième de 16 à 16+X.

Les étapes suivantes résument le fonctionnement des composants DCMI internes et donnent un Exemple de flux de données pouvant passer à travers le système DCMI:
-Extraites par l'extracteur, les données sont placées dans le FIFO de 4 mots puis ordonnées dans le registre de 32 bits comme vu plus haut.
- Après avoir reçu les différents signaux venant du bus, le synchroniseur contrôle le flux de données à travers les composants du DCMI (extracteur de données, FIFO et registre de données 32 bits).
- Une fois le bloc de données 32 bits inséré dans le registre, une demande DMA est faites.

- Le DMA transfère ensuite les données vers la destination mémoire correspondante.
La Synchronisation Hardware ce fait grâce aux signaux DCMI_VSYNC et DCMI_HSYNC :

HSYNC est utilisé pour la synchronisation de ligne. VSYNC est utilisé pour la synchronisation de frame.

Exemple avec le format JPEG :

HSYNC est utilisé pour signaler le début/fin d'un paquet. VSYNC est utilisé pour signaler la fin d'un Stream.

Le DCMI est constitué de 2 types de captures : soit une seule image soit plusieurs à la fois. C'est le DCMI_CR le registre de configuration qui permet de choisir ce mode.

En mode une seule image (snapshot mode) le bit de capture est mis à 1 par l'utilisateur ce qui démarre la capture de l'image et ce bit est automatiquement mis à 0 à la fin de la capture de l'image.

En mode plusieurs image (continous grab mode), le bit de capture est mis à 1 par l'utilisateur, ce qui démarrer l'acquisition des images. L'utilisateur doit mettre à 0 le bit de capture qui active ainsi l'acquisition.

Le DCMI supporte les formats suivants : Le format RAW, YCbCr, RGB565, JPEG.

SCCB

Le SCCB (serial camera control bus) sert à configurer un module caméra en écrivant dans différents registres. Il est un sous ensemble de l'I2C et son principe de fonctionnement est donc assez similaire à l'I2C classique. On a donc 2 fils comme pour l'I2C (3 si l'on considère la masse) : une qui donne l'horloge et l'autre qui donne les données. Lorsque des données doivent être envoyé le fil d'horloge commence à transmises. Les données sont alors envoyées et l'information est capturée sur front montant de l'horloge. La différence avec l'I2C est que l'on remplace le bit d'acknowledgement par un don't care bit. Ce bit sert à vérifier que la transmission est bien achevée.

Il y a trois types de transmission avec le sccb, une transmission pour écrire dans les registres de la caméra (à chaque fois 8 bits sont envoyés plus le don't care bit) :

-adresse (inclus ~~read~~/write bit), sous adresse, données à écrire

Une autre pour identifier une sous adresse en vue d'une lecture de registre :

-adresse (inclus ~~read~~/write bit), sous adresse

Et une pour lire un registre de donnée :

-adresse (inclus read/~~write~~ bit), donnée à lire.

La photo est ensuite acheminée par le DMA (Direct Memory Acess) à la mémoire du microcontrôleur. Puis par liaison SPI, les données sont transférées à la carte SD.

Le SPI est un protocole de transmission, composée de 4 fils :

CLK : l'horloge qui est émis par le maître lorsqu'il y a un échange de données

MOSI : Master Out/Slave In, le maître envoie des données à l'esclave

MISO : Master In/Slave Out, L'esclave envoie des données au maître

CS : Chip Select, à l'état bas lorsqu'il y a une communication (ce sera ici son seul rôle, il est plus utile dans le cas de plusieurs esclaves)

Les captures d'états se font sur le front montant de l'horloge.

C'est toujours le maître qui commence à communiquer et qui envoie un octet, si son premier bit est à 1 alors, c'est un read, il va donc écrire dans la mémoire de l'esclave. Si le premier bit est un 0 alors le maître demande à lire une donnée de la mémoire de l'esclave, une fois que le maître à finit de donner l'adresse, l'esclave lit les données inscrites à l'adresse donnée par le maître.

4-Nouvelle carte électronique

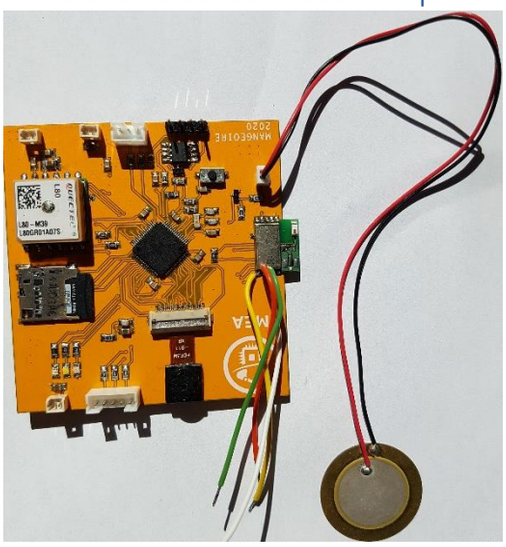


FIGURE 4 FACE DE LA CARTE ELECTRONIQUE

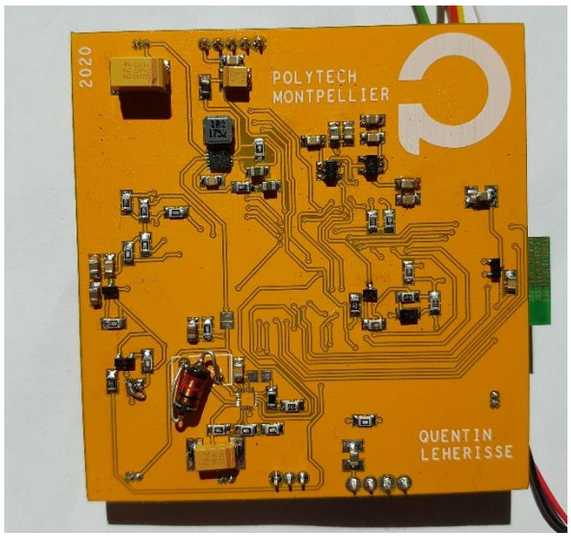


FIGURE 3 ARRIERE DE LA CARTE ELECTRONIQUE

Voici ci-dessus, un aperçu de la carte
électronique que j'ai conçu. Le défi de réalisation d'une telle carte est d'une part d'utiliser les bonnes pins pour chaque module et lors du dessin des pistes, il est important de placer les certains composants proches d'autres pour des raisons électriques (tel que les condensateurs, qwartz) mais encore réfléchir à une disposition mécanique cohérente (laisser de l'espace pour mettre la carte SD, mettre les connecteur au bord,…).

Cette carte regroupe toutes les fonctionnalités de la mangeoire :

-la partie transistor actionnable afin d'alimenter ou non un module

-La partie bluetooth, pour communiquer des informations à distance sur le stockage des données

-la partie photo

-la partie stockage

-la partie GPS afin de se repérer et récupérer l'heure

-la partie mesure du poids

-la partie alimentation batterie vers microcontrôleur

-la partie collecte d'énergie des panneaux vers batterie.

Je vais présenter point par point les différentes parties ainsi que l'OS utilisé FreeRTOS.

Description de la partie transistor

Dans une optique d'économie d'énergie, on a choisi d'alimenter les modules grâce à des transistors. Ce faisant, on peut couper l'alimentation de chaque module. Ainsi l'on peut réduire considérablement la consommation de courant chaque module consomme entre 30 et 10 mA, sachant que la consommation est du stm32 est du même ordre de grandeur que les modules.

Pour alimenter les différents modules, on a utilisé un montage semblable à celui-ci-dessous :

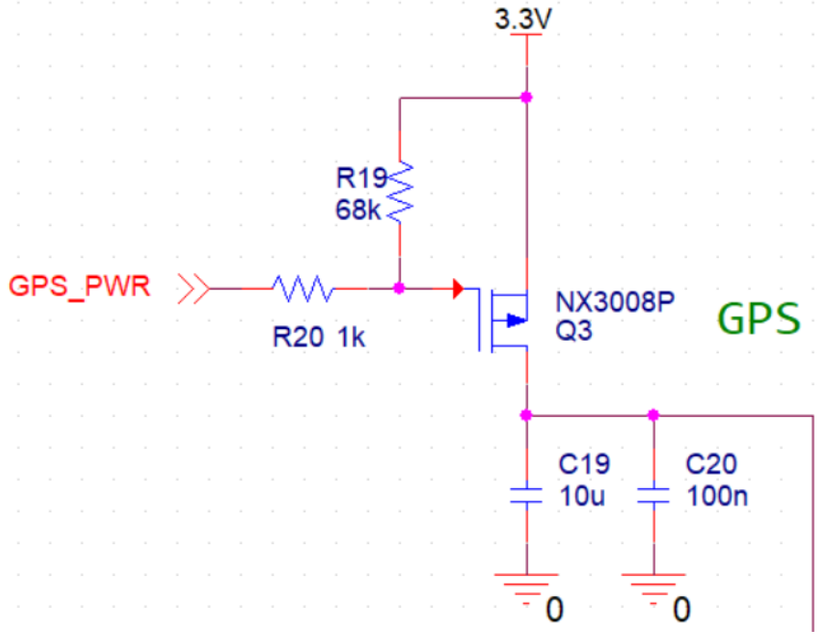


FIGURE 5 MONTAGE TRANSISTOR PNP

Il y a ici un signal venant du pin de la stm32 (GPS_PWR), et la source du transistor qui sert à alimenter la partie GPS pour ce montage. Il y a des condensateurs à la source du transistor pour linéariser la tension d'alimentation du GPS. Ce transistor est un transistor PNP, sa zone de charge espace grandit lorsqu'on fournit des charges négatives a la partie N donc il devient passant quand sa grille est à la masse. Pour économiser la charge électrique du transistor, on choisit d'utiliser un pont diviseur de tension au borne de la grille du transistor (on néglige le courant tiré par la grille), lorsque la tension n'est pas imposée par GPS_PWR, la grille est juste reliée avec le 3.3V le transistor n'est alors pas passant, lorsque GPS_PWR=0V on utilise la formule du pont diviseur de tension : Vgrille= R20/(R20+R19)*(3.3V-GPS_PWR)=1/69*3.3V (environ 0V), (Vgrille est la différence de potentiel au borne de R20).

Chaque module disposait de ce montage, voici la liste des pins associés au montages :

Le GPS => PC2

La caméra=> PC3

La carte SD => PC4

Le Bluetooth Low Energy => PC5

Le module de pesée => PA1

Afin de recevoir percevoir l'impulsion électrique émise par le piézoélectrique, on utilise le montage ci-dessous :

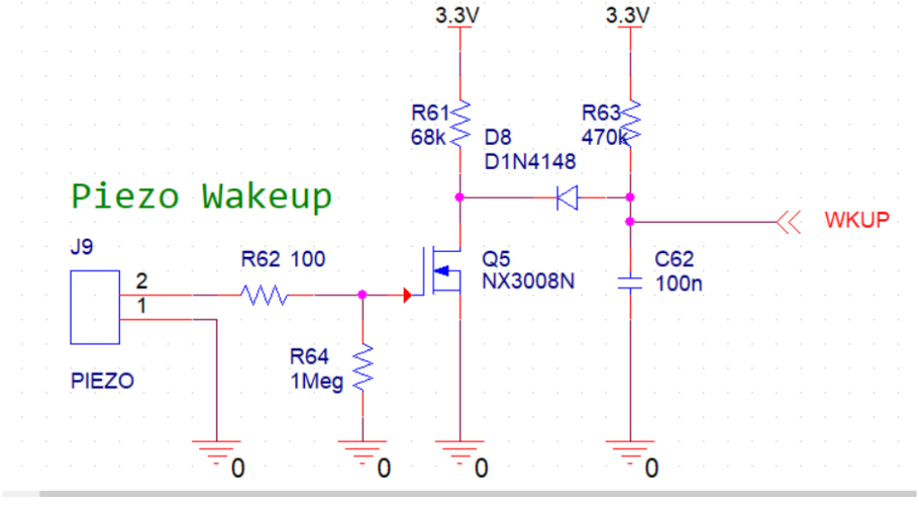


FIGURE 6 MONTAGE A TRANSISTOR NPN

Pour comprendre ce montage, il est important de comprendre comment le transistor fonctionne dans ce montage, ce transistor est un transistor NPN et est donc rendu passant grâce à un courant positif : on néglige le courant pris par la grille et j'ai mis en place un pont diviseur à la borne de la grille de tel sorte que la tension à la borne du piézoélectrique puisse être déchargé et que Vgrille = Vpiezoélectrique. Ensuite lorsque le transistor Q5 est passant on a 0V au drain. C'est alors que la diode est passante car elle a à ses bornes 0V puis 3.3V donc la capacité aux bornes de WKUP (= PC13) se vide, faisant chuter la tension de WKUP provoquant un falling edge détecté par le microprocesseur.

Description de la partie Bluetooth

On a choisi un module bluetooth afin de pouvoir transmettre via ce module les informations concernant la place restante dans la mémoire de la carte SD. Cela permet à l'utilisateur de savoir, si il lui est nécessaire de vider la carte SD.

Ce module Bluetooth low energy est le spbtle rf0 et communique avec le microprocesseur via le protocole SPI. J'utilise une librarie qui me permet de piloter le module bluetooth via des requêtes HCI, etc … par SPI.

Le code présent sur la carte peut faire marcher le module mais il ne permet pas encore de transmettre les bonnes données.

Présentation du BLE sous stm32 :

Le BLE utilise des commandes ACI (application command interface) venant d'une librarie stm32 pour faire fonctionner le module.

Ce module est implémenté selon l'architecture BLUENRG-MS.

Voici un schéma qui illustre mieux ses propos :

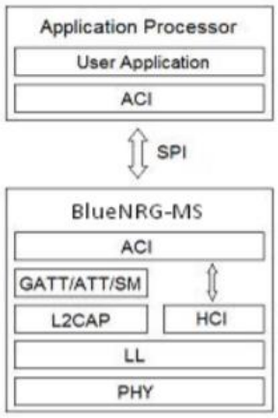


FIGURE 7 SCHEMA EXPLICATIF DES DIFFERENTES PARTIES LIE AU BLE

On voit ci-dessus que le microprocesseur parle avec le module BLE par SPI en lui envoyant des commandes ACI, l'ACI peut alors utiliser deux fonctionnalités intégrer dans le module soit le HCI (Host Command Interface) qui gère des fonctionnalité bas niveaux réglage « hardware », ou alors les différentes fonctionnalités (GATT, ATT, SM et L2CAP) qui sont plutôt des fonctionnalités haut-niveaux pour la « communication ».

Enfin il reste deux « fonctionnalités » dans le module qui sont celle qui gère la couche physique émission sur certaines fréquences propre au bluetooth ou le LL (Link Layer) couche de liaison qui gère les états du BLE écoute, recherche, connexion, émission… et qui guide l'envoie des données par paquets propres au bluetooth.

Couche physique :

Le BLE utilise une fréquence pour ses ondes de 2.400 à 2.483 GHz. Chacune est espacée de 2MHz, ce qui nous fait un total de 40 channels.

3 de ces channels sont utilisés pour l'advertising, c'est-à-dire pour partager des données avec n'importe quel module, ceci comprend aussi les paquets(packets) pour détecter et connaitre les états des autres modules.

Couche de liaison :

La couche de liaison peut être représenté comme une alternance de 5 états :

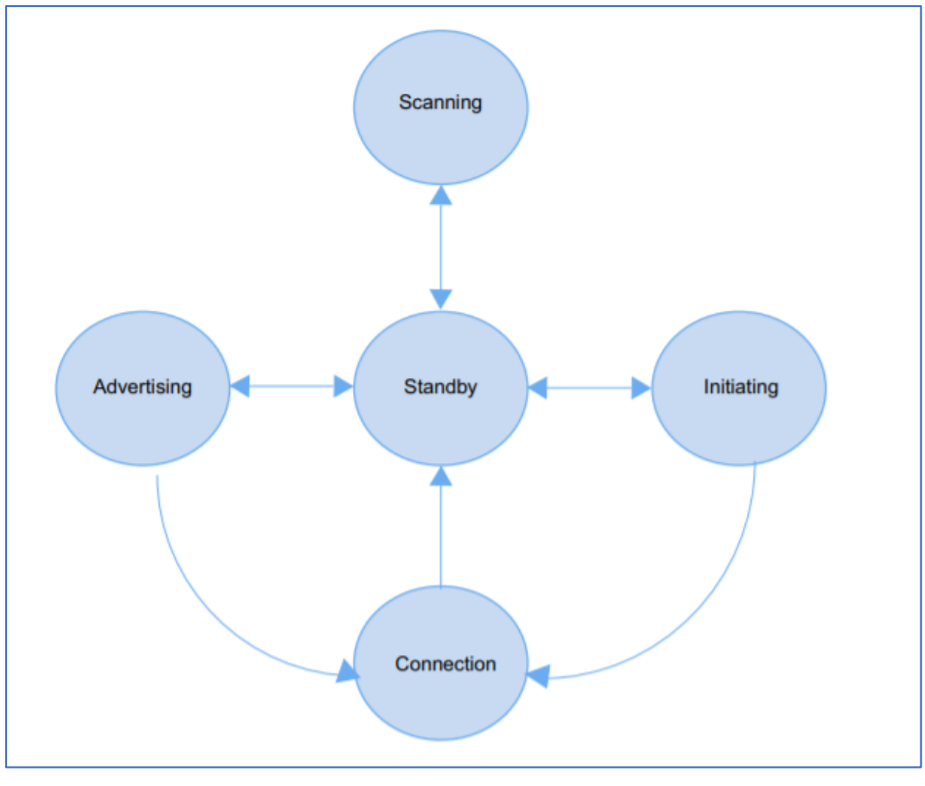


FIGURE 8 DIAGRAMME DES ETATS DE COUCHE DE LIAISON

Le mode scanning : il sert à détecter d'autres modules en mode advertising.

Le mode standby :il n'envoie ou ne reçoit aucun paquet.

Le mode advertising : il émet des paquets sur les channels d'advertising.

Le mode initiating : le module débute une transmission avec un module en mode advertising.

Le mode connection : Le mode en mode initiating est en rôle maître (master), il communique avec le module en mode esclave(slave) et définit les timings de transmissions. Le module en mode slave était auparavant en mode advertising, désormais il communique juste avec son maître jusqu'à ce que la transmission soit finie.

On note que notre module est en mode périphérique, c'est-à-dire qu'il est toujours l'esclave, c'est à lui de fournir les données.

La composition des paquets :

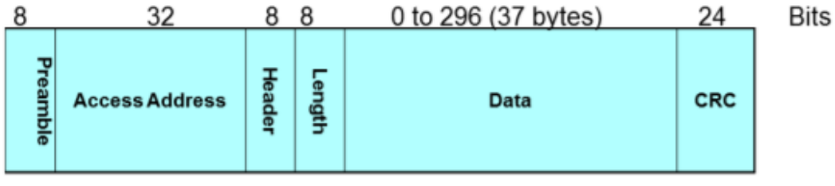


FIGURE 9 COMPOSITION DES PAQUETS

La partie Host Controller Interface (HCI):

Ici le HCI permet au contrôleur et à l'hôte de communiquer ensemble, ici via SPI.

La partie Logical link control and adaptation layer protocol (L2CAP) :

Le protocole de couche de contrôle et d'adaptation de liaison logique (L2CAP), s'occupe de la segmentation de paquets, des opérations de réassemblage, du multiplexage des différents protocoles et le transport de l'information sur la qualité de service. La couche L2CAP permet entre autres la mise à jour des paramètres de connexion en mode esclave.

La partie Security Manager (SM) :

Il sert à encrypter et authentifier les messages qui circulent par bluetooth, notamment grâce au Cipher Block Chaining-Message Authentication Code (CCM) algorithm et au 128-bit AES block cipher.

La partie attribute protocol :

Le protocole ATT (Attribute Protocol) permet à un module bluetooth d'exposer certaines données, appelées attributs, à un autre module bluetooth. Le module qui expose ses attributs est appelé le serveur (host). Le périphérique qui les reçoit, est appelé le client.

La partie Generic attribute (GATT) profile :

Le profil d'attribut générique (GATT) définit un cadre d'utilisation du protocole ATT.

Note sur le codage :

Les interruptions et les timers utilisés par le code stm32 ont été adaptées pour fonctionner sous FreeRTOS. En effet certains timers étaient utilisés pour compter des choses différentes, c'est pour cela que j'ai configuré d'autres timers pour compter ce qui est nécessaire pour le BLE. Pour les interruptions, les priorités de la plus importante à la priorité 4 sont toutes réservées pour FreeRTOS, il faut donc changer les priorités du code BLE en conséquence. Exemple :

```
void TIM2_IRQHandler()
{
    TIM2->SR &= ~TIM_SR_UIF;
    BSP_IncTick();

}
```

FIGURE 10 CODE EXECUTE AU DECLENCHEMENT DE L'INTERRUPTION DE TIM2

Cette fonction ci-dessus est une fonction déclenchée par le timer 2 cette interruption était dans la version initiale du code BLE mais l'interruption venait du timer systick.

Pour savoir quand transmettre les données par bluetooth, il faut regarder les différents modes par lesquels le bluetooth entre. On observe lorsque que c'est lorsque le module entre en mode advertising, il entre en mode slave et informe le master. Au niveau du code cela se traduit par la modification de l'intérieur d'un des « cases » de la fonction « user_notify », fonction qui gère les commandes et fonction à exécuter selon les différents modes de fonctionnement.

Description de la partie Photo

La partie photo a été déjà traitée par une des équipes précédentes, mon travail dessus a donc été de vérifié que la capture d'image soit faite rigoureusement.

Nous expliquerons ici juste le principe de la compression JPEG. La compression est utilisée sur les captures du module Photo.

Voici le schéma qui explique les différentes parties :

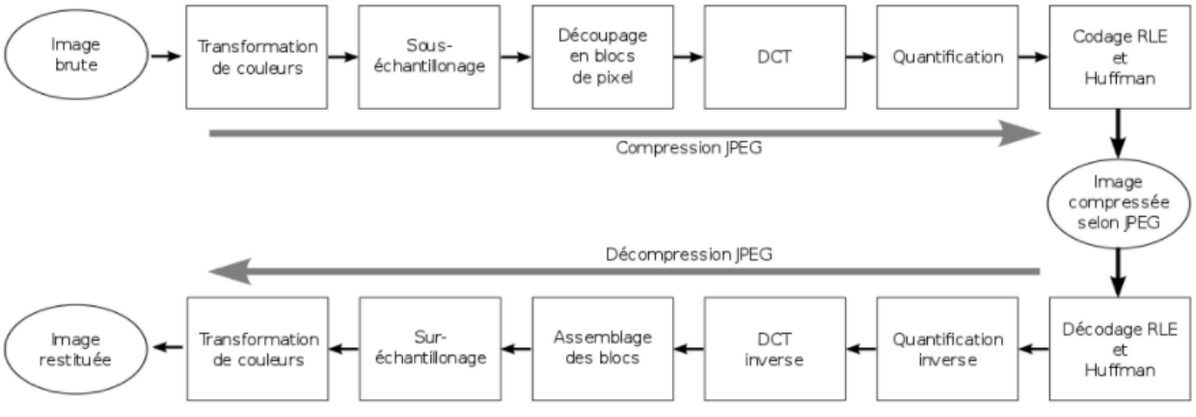


FIGURE 11 ETAPES COMPRESSION/DECOMPRESSION JPEG

La compression JPEG est composée de plusieurs 6 étapes qui vont être expliqué par la suite, le chemin inverse c'est-à-dire la décompression JPEG sera aussi expliqué.

Ainsi à chaque bloc de la compression est associé un bloc dans la décompression qui vient défaire le travail effectué par le codage JPEG.

Transformation de couleurs :

La transformation de couleur est le passage du modèle RVB(RGB) au modèle YCbCr pour la compression ou inversement pour la décompression.

Le modèle RGB repose sur l'addition du Rouge, Vert, Bleu pour recréer une couleur.

Le modèle YCbCr repose aussi sur trois variables Y la luminance et deux chrominance Cb et Cr.

Le ' ici sera utilisé pour désigner des grandeurs relatives.

On utilise la formule suivante pour convertir RVB en YCbCr :

Y'=0,299R'+0,587V'+0,114B'

Cb=-0,1687R'-0,3313V'+0,5B'+128

Cr=0,5R'-0,4187V'-0,0813B'+128

La conversion inverse se fait comme ceci :

R=Y'+1,402(Cr-128)

V=Y'-0,34414(Cb-128) -0,71414(Cr-128)

B = Y' + 1,772 (Cb - 128)

Sous/Sur échantillonnage :

Ici on sous-échantillonne la chrominance, c'est-à-dire on réduit le nombre de Cr et Cb.

Pour le sur-échantillonnage, on augmente ce nombre pour arriver à son niveau d'origine.

Découpage/Assemblage en bloc :

Pour le découpage, on coupe l'image en groupe de 8*8 pixels. Chaque bloc est alors traité séparément.

Pour l'Assemblage, les blocs sont réassemblés.

DCT : pour transformée en cosinus discret

A chacun des blocs, on aplique une transformation en cosinus discrète.

Pour la transformée en cosinus inverse, on applique une transformation en cosinus inverse.

Quantification : on divise les matrices du DCT par une autre matrice appelée matrice de quantification.

L'effet de cette division, réduit la différence entre entre les coefficients d'une même matrice

Pour la quantification inverse, il suffit de diviser par la matrice de quantification.

Le codage RLE : consiste à indiquer pour chaque suite de pixels d'une même couleur le nombre de pixels de cette séquence. Tel que AAABBBBBBBBBjjjj ->A3B8j4

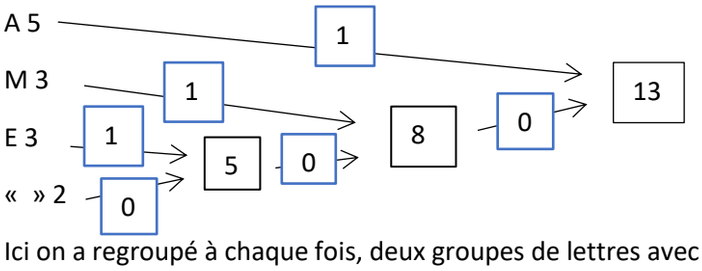
Pour le décodage RLE, on utilise le procédé inverse : A3B8j4->AAABBBBjjjj

Le codage de Huffman : un algorithme de compression sans perte

C'est-à-dire que l'on crée un dictionnaire de signe représenté par des 1 ou/et des 0 puis l'on se sert de ce dictionnaire pour coder un mot, Exemple :

Au lieu de coder les caractères avec un code ascii (8 caractères), ainsi en code ascii MEA MEA MEAAA se code :
0100110101000101010000010010000000100110101000101010000001001000000100110101000101010000010100000101000001

On va créer un dictionnaire puis s'en servir pour recoder le message :



Ici on a regroupé à chaque fois, deux groupes de lettres avec les nombres les plus faibles.

Ensuite on met un 1 aux flèches qui viennent du dessus et 0 à celle du dessous.

Enfin on compose le dictionnaire à l'aide du schéma. On part du plus gros nombre vers les lettres en récoltant les 1 et 0 au passage.

Ainsi on obtient un dictionnaire tel que :

A=1, M=01, E=001, « »= 000

On peut coder MEA MEA MEAAA en 010011000010011000010011111.On observe un énorme gain de stockage en utilisant le codage Huffman.

Pour le décodage il suffit de lire le dictionnaire dans le sens inverse.

Description de la partie stockage

On a choisi une carte micro SD car c'est une version compacte de la carte SD, c'est également une carte dont il est facile de récupérer ses données puisqu'il suffit d'utiliser un adaptateur qui se glisse dans l'ordinateur.

La partie stockage a également été traitée par la même équipe que pour la partie photo, j'ai vérifié son bon fonctionnement et j'ai ajouté un fichier qui lie date et la position GPS.

Pour stocker les informations, dans la carte on utilise simplement la fonction f_write,

Exemple :

```
fresult = f_open((FIL *)&myfile, (TCHAR *)filename, (BYTE) FA_CREATE_ALWAYS | FA_WRITE);

if (fresult == FR_OK)
{
    my_printf("Image file ready !\r\n", (char*)filename);
}

else
{
    // File creation failed
    my_printf("Image opening failed\r\n");
    BSP_LED_On(RED);

    // Catch application here until Watchdog resets
    while(1);
}

uint16_t nb_cluster = 0;
nb_cluster = 40000/128;
for(uint32_t c = 0; c < nb_cluster; c++){
    fresult = f_write(&myfile, &dcmi_dma_buffer_HQ[c*128], 512, NULL);
}
fresult = f_write(&myfile, &dcmi_dma_buffer_HQ[39936], 128, NULL);
fresult = f_close(&myfile);
```

FIGURE 12 CODE ECRITURE EN MEMOIRE

Ici après avoir ouvert le fichier de la carte SD grâce à f_open, on écrit dans la carte sd, le contenu du buffer du DCMI/DMA. Enfin on ferme le fichier avec f_close.

Description de la partie GPS

On a choisi d'intégré un GPS afin qu'il puisse récupérer à chaque démarrage l'heure, la date et la localisation.

Le dispositif que j'ai utilisé est le L80-M39, c'est un module GPS une fois alimenté, fonctionne sans aide pour trouver les satellites nécessaires à la localisation. Il communique grâce à une liaison USART : PB 10 GPS RX/USART RX, PB11 GPS TX, USART RX. Il transmet des trames NMEA par USART. Il convient donc de créer un parser permettant de récupérer les informations qui nous intéresses tel que la date, l'heure et la localisation.

Voici un exemple de quelques trames NMEA :

$GPGSA,A,3,11,08,01,03,,,,,,,,2.56,2.36,1.00*0F

$GPRMC,160754.000,A,4826.8746,N,00251.5236,W,0.31,205.22,020620,,,A*7F

$GPVTG,205.22,T,,M,0.31,N,0.58,K,A*35

$GPGGA,160755.000,4826.8746,N,00251.5234,W,1,4,2.36,102.6,M,49.5,M,,*43

$GPGSA,A,3,11,08,01,03,,,,,,,,2.56,2.36,1.00*0F

$GPRMC,160755.000,A,4826.8746,N,00251.5234,W,0.18,141.07,020620,,,A*73

$GPVTG,141.07,T,,M,0.18,N,0.33,K,A*37

$GPGGA,160756.000,4826.8747,N,00251.5235,W,1,4,2.36,102.7,M,49.5,M,,*41

$GPGSA,A,3,11,08,01,03,,,,,,,,2.56,2.36,1.00*0F

$GPGSV,2,1,07,08,84,149,38,11,65,288,32,01,40,264,31,20,10,046,*78

On peut voir que la trame commence toujours par $ puis une suite de 5 lettres, ces 5 lettres nous permettent de connaître le contenu du reste de la phrase, vient ensuite * puis le CRC, le CRC est un XOR entre tous les caractères, par exemple :

$GPRMC,160754.000,A,4826.8746,N,00251.5236,W,0.31,205.22,020620,,,A*7F

Cette phrase débute par GPRMC, GP pour pour Global Positioning System et RMC pour données minimales exploitables spécifiques

160754=Heure du Fix 16:07:54 UTC

4826.45,N = Latitude 48 deg. 26.8746 min North

00251.5236,W = Longitude 002 deg. 51.5236 min West

020620 = Date du fix 02 Mai 2020

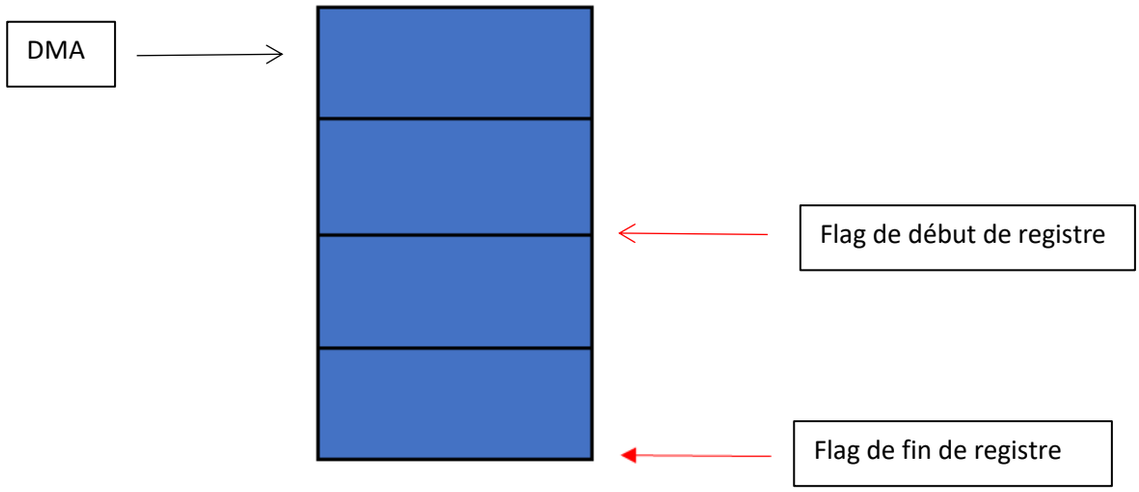
*7F = checksum obligatoire

Les autres valeurs dans cette phrase sont pour nous inutile.

J'ai donc réalisé un parser dans un premier temps les caractères venant de l'USART sont stockés dans un buffer grâce à la fonction DMA (2 flags s'allume pour signaler au programme si c'est la première moitié ou la seconde qui est pleine et l'autre est donc à ce faire remplir). Exemple :

On représente le buffer par 4 registres pour schématiser.

Supposons que la DMA est représentée par une flèche noire qui parcours continuellement le buffer afin de remplir chacun de ses registres et les deux flèches rouges sont les flags qui sont mis à 1 à chaque fois que la flèche DMA passe là où le flag est positionné.



Ceci fait j'analyse les caractères du buffer afin de trouver le début d'une phrase NMEA (débute par $), ensuite je récolte tous les caractères suivants jusqu'au CRC, j'effectue un contrôle pour voir si le CRC colle bien avec la phrase envoyée. Ensuite je découpe la phrase en mot, grâce aux virgules entre chaque mot des trames NMEA. Les différents mots sont rangés dans un tableau. Ensuite selon la composition du premier mot de la trame, on peut savoir quelle information est contenu dans la phrase.

Il ne me reste plus qu'à copié ses informations dans mes registres :

```
if ((nmea_field[1][0]>=0x30) && (nmea_field[1][0]<=0x39))
{
    hours =   (nmea_field[1][0]-48) * 10 + (nmea_field[1][1]-48);
    minutes = (nmea_field[1][2]-48) * 10 + (nmea_field[1][3]-48);
    seconds = (nmea_field[1][4]-48) * 10 + (nmea_field[1][5]-48);
}
else
{
    hours =   0;
    minutes = 0;
    seconds = 0;
}
```

FIGURE 12 CODE DE RECUPERATION DE L'INFORMATION

Les informations stockées dans les tableaux sont des caractères, il faut donc leur retirer la valeur '0' ou 48 avant de pouvoir les lires. Les champs des différentes phrases ne changent jamais de contenu c'est pourquoi, il est facile de trouver l'information dans le tableau : il suffit de trouver où sont les caractères désirés puis de recopier les valeurs dans un registre.

L'heure sert donc à mettre à jour la RTC. La date sert donc aussi à mettre à jour la RTC. La localisation quant à elle est stockée dans la carte SD.

Description de la partie mesure du poids

Le poids est mesuré grâce à la variation de résistance d'un pont de wheatstone face à un poids.

Nous avons ensuite ce montage ci -dessous qui prend la tension aux bornes du pont de wheatstone :

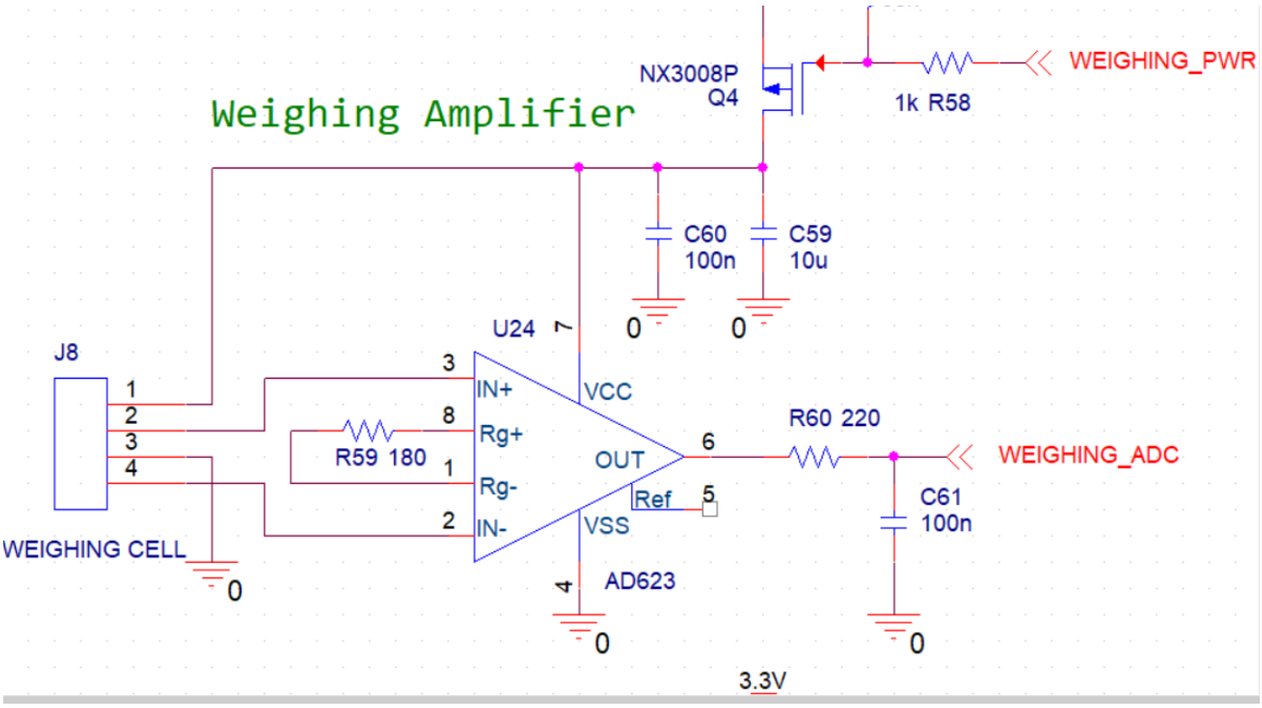


FIGURE 13 AMPLIFICATEUR DU PONT WHEATSTONE

C'est donc la différence entre IN+ et IN- qui est mesuré par l'ampli AD623.

Cet ampli va sortir une tension proportionnelle à la différence de tension entre IN+ et IN-. On a mis en sortie de l'ampli un filtre passe bas pour linéariser la tension de sortie de l'ampli. On a mis une résistance faible pour R59 afin d'avoir un gain élevé.

Selon la datasheet la formule du gain est R59=100k/(G-1)

On veut déterminer G :

R59*G-R59=100k

R59*G=100k+R59

G=(100k+180) /180, R59=180 Ohm

G≃556.

La pulsation de coupure d'un filtre passe bas est de 1/(R*C), ici R=R60 et C=C61.

Donc la fréquence de coupure est 1/(2*pi*R*C) :

$$1 \div (2 \times \pi \times 220 \times 100 \times 10^{-9})$$

≃7234Hz.

La tension en sortie du filtre passe bas est récupéré par le microcontrôleur par le pin PA0.

On configure donc une ADC sur le pin PA0 afin de récupérer le signal de pesée.

Description de la partie alimentation batterie vers microcontrôleur

La tension de la batterie est une tension variable en fonction de son état de charge, et sa valeur nominale est différente de la tension de fonctionnement c'est pour cela que nous avons besoin d'un composant qui permet de ramener la tension de la batterie à du 3.3V.

Le choix s'est donc porté sur un module convertisseur buckboost, c'est-à-dire un convertisseur de tension capable d'augmenter ou diminuer la tension à sa propre sortie. Le nom du module choisi est le TPS63060.

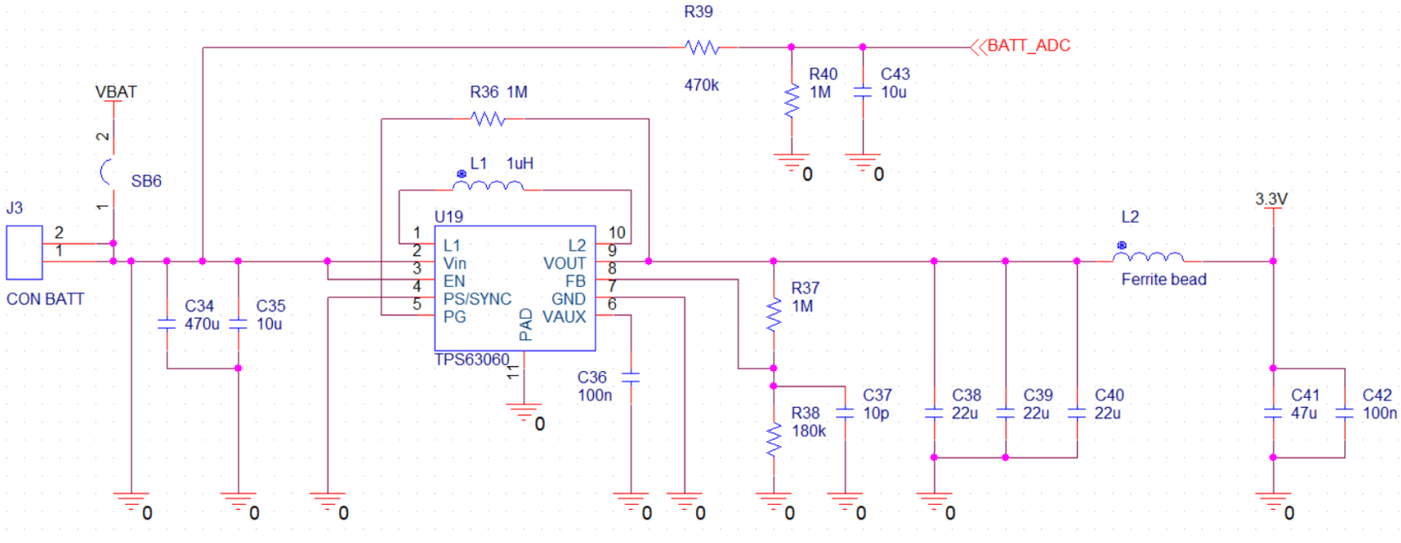


FIGURE 14 ALIMENTATION BATTERIE VERS MICROCONTROLEUR

Ce module sert à transformer la tension de la batterie en une tension 3.3V exploitable par le microcontrôleur.

On utilise des condensateurs pour linéariser la tension en entrée et en sortie.

La datasheet nous dit que la valeur typique de FB est de 500mV.

On nous dit aussi de garder la résistance R38 proche de 200k Ohm.

Et nous avons cette équation :

$$R1 = R2 \times \left(\frac{V_{OUT}}{V_{FB}} - 1 \right)$$

On choisit donc une résistance R38 à 180k. On applique la formule :

180000*(3.3/0.5-1) =1008000 donc ≃ 1M Ohm.

Sur le schéma de la carte ci-dessus on peut voir qu'il y a un fil : BATT_ADC se fil est relié à un pin, le pin PA5. On peut utiliser PA5 afin d'observer le niveau de tension de la batterie. Ainsi en déduire sont état de charge car le niveau tension varie légèrement en fonction de son état de charge plus elle est chargée plus la tension est élevée, et inversement plus elle est déchargée plus la tension baisse.

Description de la partie collecte d'énergie des panneaux vers batterie

La partie stockage a été traitée par la première équipe que pour la partie collecte d'énergie avec panneaux solaires, j'ai vérifié son bon fonctionnement.

Description de la Partie FreeRTOS et comportement du module

J'ai dû coder des fonctions pour chaque module ainsi mais certains modules peuvent nécessiter parfois de fonctionner en même temps, c'est pourquoi j'ai intégré FreeRTOS à mon programme.

Le programme commence donc par régler son horloge grâce à la fonction SystemClock_Config. L'horloge se calibre pour fonctionner à 80MHz.

Ensuite il crée toutes ses tâches qu'il va exécuter, on lance le taskscheduler et ça y est nous avons lancer nos tâches ! La première tâche qui se lance est celle qui vérifie les flags de la clock et de l'initialisation de la clock pour savoir quelle tâche lancer.

Soit le module démarre et doit s'initialisé, il va alors utiliser le GPS pour trouver sa date, son heure et sa position puis lance une première fois le Bluetooth.

Sinon il se réveille, soit il est réveillé par une impulsion du piézoélectrique, le flag de la clock ne sera pas allumé alors la prise de la photo se lancera.

Puis si c'est le minuteur qui réveille la mangeoire se sera la tache Bluetooth qui s'exécutera.

Enfin à la fin des tâches Bluetooth ou prise de photo, le module se replonge en veille.

Contrainte software :

A ce niveau de programmation en logiciel temps réel, la difficulté de programmation vient de la taille de mémoire : il faut vérifier que chaque tâche n'a pas trop de mémoire alloué pour en laisser aux autres fonctions du microcontrôleur. Il faut aussi optimiser les différents programmes en termes de mémoire, faire des programmes qui fonctionne avec des buffer plus petit, etc…

Il faut aussi les rendre suffisamment efficace afin qu'il puisse respecter les contraintes de temps réelle, (exemple : exécuté la prise de photo suffisamment rapidement pour pouvoir pesée l'oiseau avant qu'il ne reparte).

Travail restant :

Il reste encore à faire fonctionner la prise de photo lorsque le Bluetooth est en fonctionnement.

La durée du fonctionnement Bluetooth sera limitée par la capacité de la batterie : plus la batterie est vidée plus le fonctionnement du Bluetooth devra être interrompu.

Enfin il reste à transmettre les informations venant de la carte SD car le bluetooth fonctionne mais il ne transmet que des informations venant d'une démo.

5 Manuel d'utilisation

Ce manuel est à l'attention des futurs utilisateurs de cette mangeoire connectée.

Pour utiliser cet appareil, il faut placer le placer dans un endroit accessible aux oiseaux. Puis il faut placer le piézoélectrique au point d'atterrissage de l'oiseau. Il faut mettre le panneau solaire en évidence face au soleil. Faites-en sorte que l'objectif photo soit positionné de manière à photographier l'oiseau au point d'atterrissage.

Vous pouvez ensuite connecter le panneau solaire à la mangeoire.

Enfin vous connecter la batterie. Le module va démarrer, il faudra lui laisser 30 secondes à quelques minutes afin qu'il puisse récupérer l'heure, la date et sa position.

Puis le bluetooth démarre, votre mangeoire est à présent opérationnelle et photographiera les oiseaux qui se poseront sur cette mangeoire.

Ci-dessous une indication des branchements de la mangeoire :

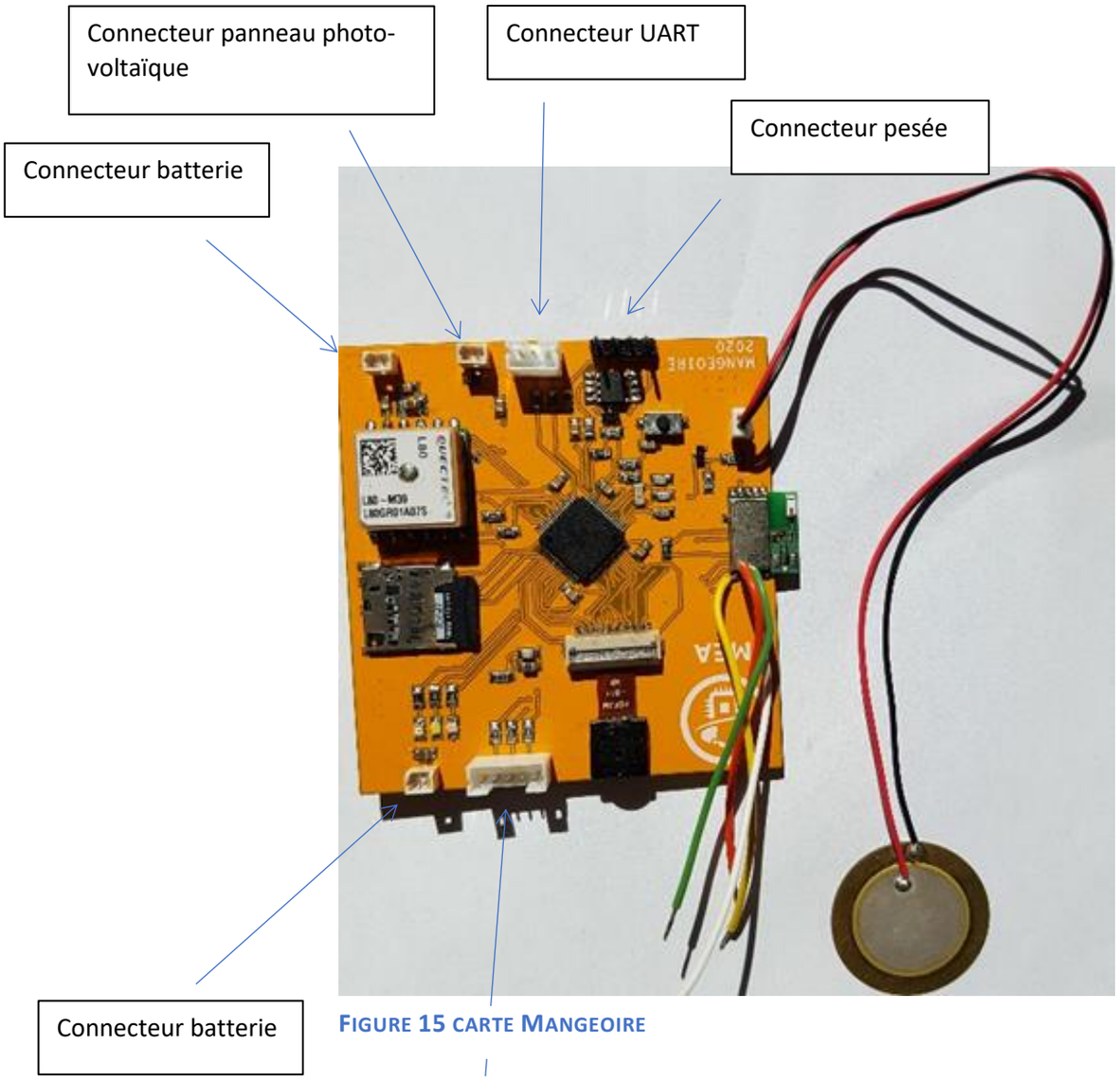


FIGURE 15 CARTE MANGEOIRE

6-DDRS (Développement durable et responsabilité sociale)

Quelle est la position de l'entreprise vis à vis des enjeux du DD RS ?

Ayant fait mon stage dans les locaux de Polytech je suis plus à même de parler des différents enjeux de Polytech vis-à-vis du DDRS. Le DDRS est profondément inscrit dans les valeurs de Polytech. C'est une mission qui vient en support de la formation d'ingénieur que délivre Polytech.

Quels sont les leviers qui motivent l'intérêt de l'entreprise pour le DD RS ?

Polytech à reçu un label DDRS pour son implication, c'est l'un des 10 premiers établissements à recevoir cette distinction, cela l'incite à continuer dans ces efforts.

Quelle est l'organisation mise en place pour répondre à ces questions ?

Il existe des associations d'élèves tel que Polyearth qui agissent dans une optique de développement durable, l'administration est aussi très concernée et fait office de moteur pour tous les projets DDRS.

Quelles sont les actions mises en œuvre ?

La mission de Polytech est de former des ingénieurs, sous statut d'étudiants ou d'apprentis en entretenant et développant des relations avec les acteurs sociaux-économiques aux niveaux national et international. L'ingénieur doit aujourd'hui assumer une double responsabilité dans la société en maitrisant des techniques de plus en plus évoluées au service de la communauté mais également en maitrisant les risques globaux qu'elles engendrent. Polytech s'engage pour des causes tel que le téléthon et permet à nombre d'intervenant de s'exprimer sur les sujets comme la responsabilité sociétale et environnementale afin de sensibiliser les polytechniciens.

7-Conclusion

Conclusion professionnelle

J'ai réalisé un projet avec une méthode scientifique. Au travers de ce projet, j'ai pu développer mes compétences en programmation C et en FreeRTOS. J'ai également acquis de solides connaissances sur le test unitaire. J'ai pu intégrer de quelques dispositifs à cette mangeoire et tester leurs bons fonctionnements sur une carte.

Retour sur expérience personnel

Ce stage m'a permis de m'améliorer dans la programmation et la conception de carte électronique. Surtout j'ai eu l'occasion de découvrir l'environnement startup. Ma vision de ma future carrière en tant qu'ingénieur à évolué. À la lumière de cette expérience, je souhaite me tourner vers la partie vers la conception de systèmes embarqués.

Merci pour votre attention !

Source :

https://www.st.com/resource/en/application_note/dm00373474-digital-camera-interface-dcmi-on-stm32-mcus-stmicroelectronics.pdf

http://caxapa.ru/thumbs/799244/MIPI_Alliance_Specification_for_Camera_S.pdf

https://www.nxp.com/docs/en/application-note/AN5305.pdf

https://www.st.com/resource/en/programming_manual/dm00141271-bluenrg-bluenrgms-stacks-programming-guidelines-stmicroelectronics.pdf

https://www.st.com/resource/en/user_manual/dm00162667-the-bluenrgms-bluetooth-le-stack-application-command-interface-aci-stmicroelectronics.pdf

https://fr.wikipedia.org/wiki/YCbCr#:~:text=Le%20mod%C3%A8le%20YCbCr%20ou%20plus,ou%20en%20noir%20et%20blanc.

https://objectif-languedoc-roussillon.latribune.fr/innovation/2019-10-10/la-satt-axlr-prepare-un-accelerateur-de-deep-techs-830357.html

Annexe :

MIPI CSI2

Le MIPI (ou Mobile Industry Processor Interface) alliance est un groupe qui crée des interfaces et autres produits utilisées dans l'industrie informatique mobile. Elle a donc créé une interface qui fait la liaison entre processeur et caméra notamment utilisé pour les raspberry pi : la CSI (Caméra Serial Interface), il y a 3 versions de la CSI de la moins rapide et plus ancienne (environ 1 Gbits/s) à la plus récente et plus performante (environ 5 Gbits/s). Le CSI2 est très utilisé pour beaucoup de modules caméra notamment dans la plupart des smartphones. Il est donc très populaire. Notamment car les nouveaux modules caméra nécessite un débit assez élevé qui peut être fournir par l'interface CSI2.

Voici un schéma représentant les différentes parties du protocole MIPI CSI2:

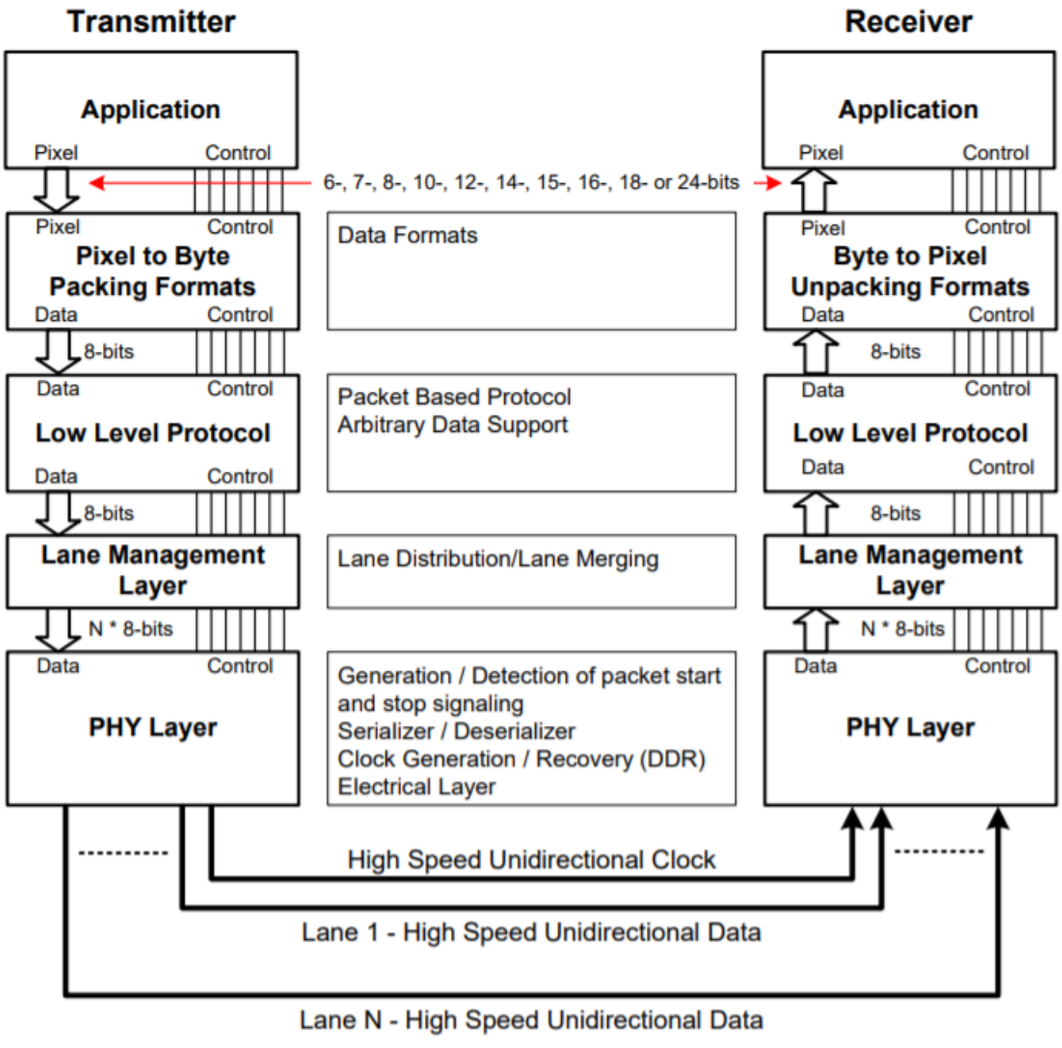


Figure 1schéma des différentes couches du CSI2

Le **phy layer** est la partie qui transmet les données physiquement. Il coordonne les bits en fonction de l'horloge conduite vers le récepteur phy layer.

Pixel/Byte packing unpacking layer, ce layer utilise entre 6 et 24 bits pour coder un pixel, il reçoie ou transmet les données par paquet de 8 bits.

Le **low level protocol** lui s'occupe de transformer la série de bit en paquet, il ajoute le start of transmission (signale le début de la transmission) et le end of transmission (signale la fin de la transmission).

Le schéma ci-dessous illustre une communication typique :

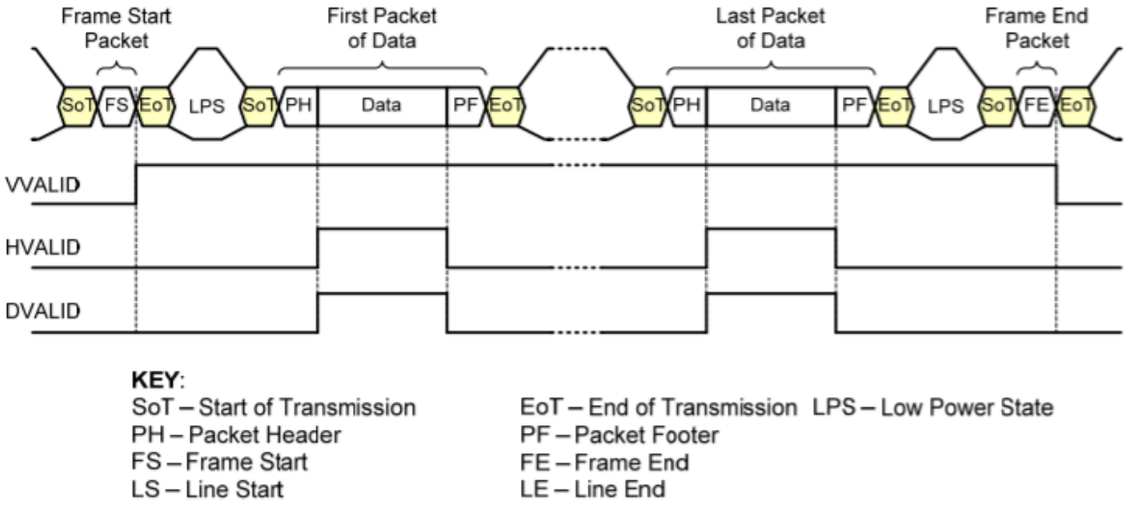


Figure 2 présentation des 2 types de paquet

Il y a deux types de paquet définit : les longs et les courts. Les paquets sont représenté ci-dessus : FS et FE sont des paquets courts, la data est envoyé en paquet long.

Un paquet long est composé d'un paquet Header de 32 bits, une application specific Data Payload avec un nombre variable d'octet et un Packet footer de 16 bits.

Le paquet header est composé d'un data identifier qui donne des informations sur le contenu du paquet, le word count de 16 bits pour déterminer la fin du paquet et 8 bits de détection d'erreur.

Le packet data ou application specific payload contient le nombre d'octet écrit dans le word count.

Le paquet footer est un checksum de 16 bits.

Un paquet court n'est composé que d'un paquet header le word count est remplacé par de la data.

Le Data identifier lui est composé d'un virtual channel identifier et un data type. Une virtual channel est associée à un flux de donnée indépedant. Le virtual channel identifier est composé de 2 bits qui donnent l'information du flux de donnée qui va être emprunté par le paquet.

Ces flux de données sont des vagues de data, schéma pour 2 channels :

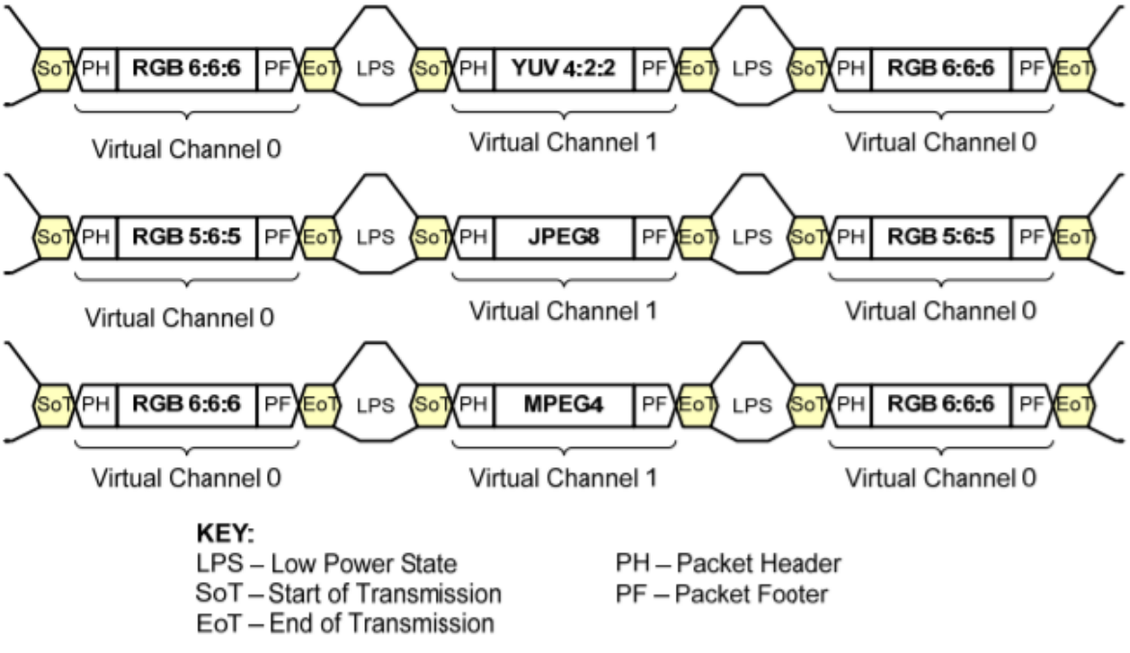


Figure 3 vagues de data

Le Data type lui identifie les classes de data voici un tableau qui décrit les différents types de data :

Data Type	Description
0x00 to 0x07	Synchronization Short Packet Data Types
0x08 to 0x0F	Generic Short Packet Data Types
0x10 to 0x17	Generic Long Packet Data Types
0x18 to 0x1F	YUV Data
0x20 to 0x27	RGB Data
0x28 to 0x2F	RAW Data
0x30 to 0x37	User Defined Byte-based Data
0x38 to 0x3F	Reserved

Figure 4 tableau des data types

Note : YUV, RAW, RGB sont des formats d'images.

Le **lane management layer** distribue les différents bits sur les différentes lignes HSUD. Voici un schéma qui représente cette distribution :

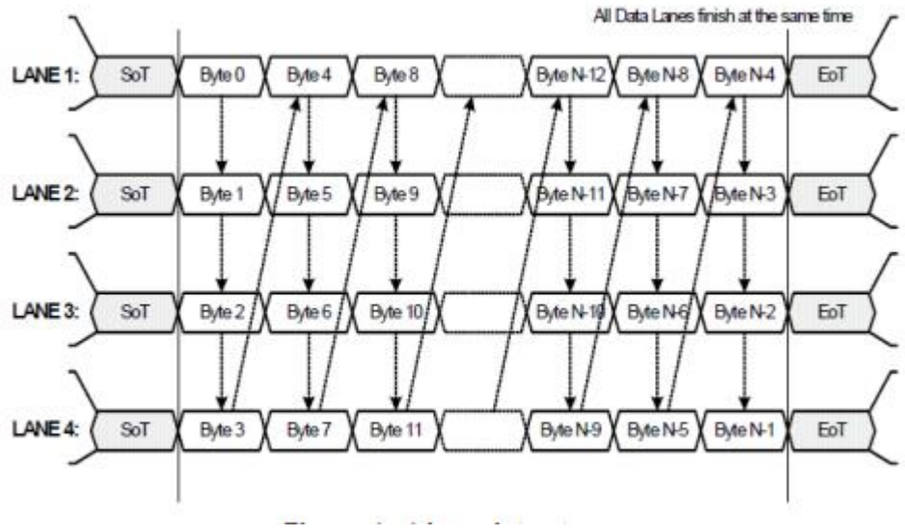


Figure 5 insertion des bits sur les différentes lignes

Le CSI2 comprend aussi le **camera control interface/CCI**, il sert à contrôler le transmetteur ici une caméra, le CCI est un sous ensemble de l'I2C, il supporte des opérations à 400kHz. Le receveur est le maître et le transmetteur est l'esclave.

Un message CCI basique est constitué de la start condition, l'adresse de l'esclave avec le bit de lecture écriture, acknowledge, on peut ajouter une sous adresse encore un acknowledge puis la data avec acknowledge ou non (donnée par le récepteur ou le transmetteur selon l'opération), (de la data et acknowledge si besoin) et enfin un bit de STOP condition.