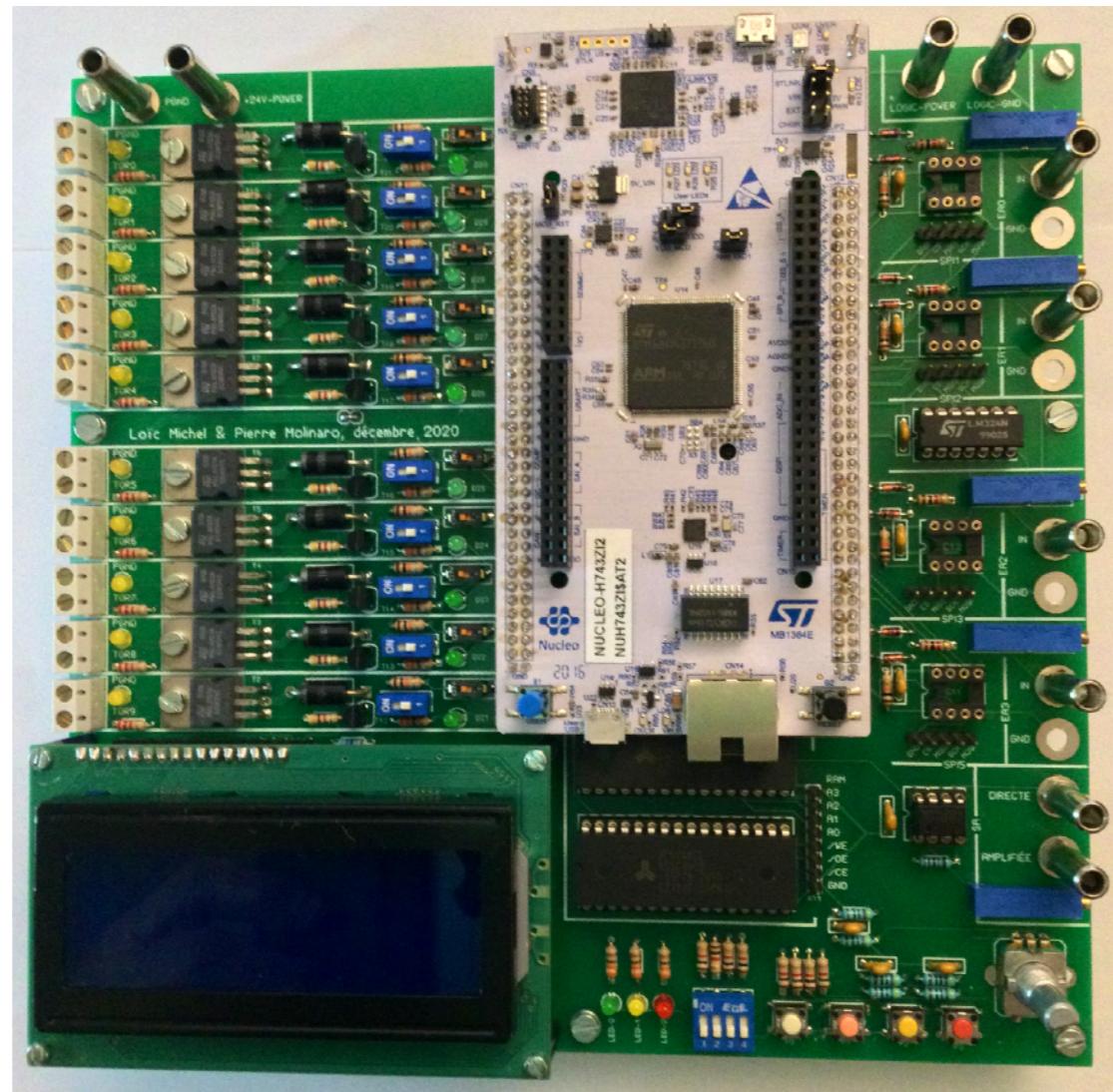


Mode d'emploi de la carte

STM32H743 LHEEA



Pierre Molinaro

Janvier 2021

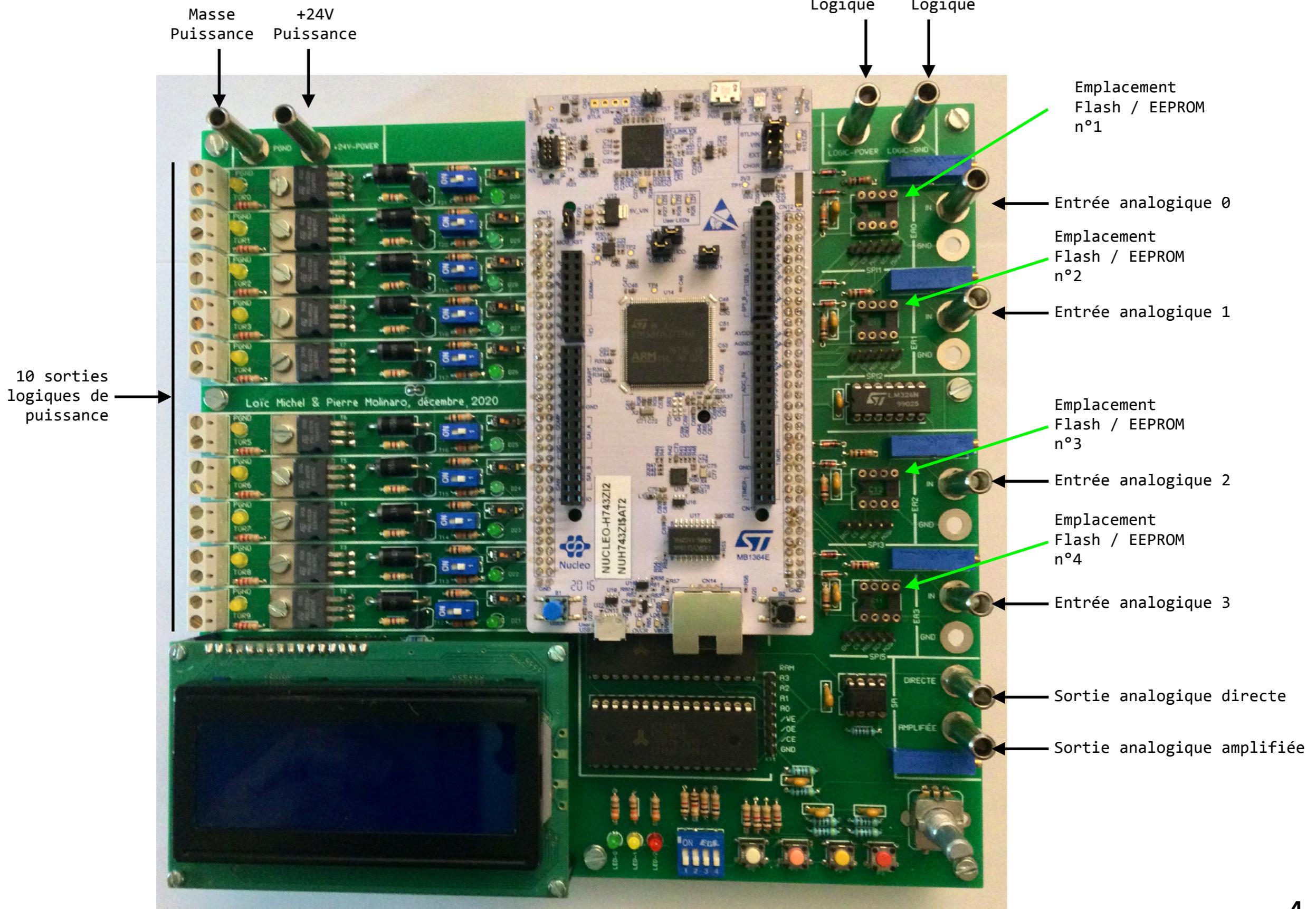
Introduction

Carte STM32H743 LHEEA

Le fichier *EICanari* de la carte, les fichiers *keynote* et *pdf*, les croquis d'exemple, sont disponibles à l'URL :
<https://github.com/pierremolinaro/cartes-micro-controleurs-centrale-nantes/tree/master/carte-h743-lheea>

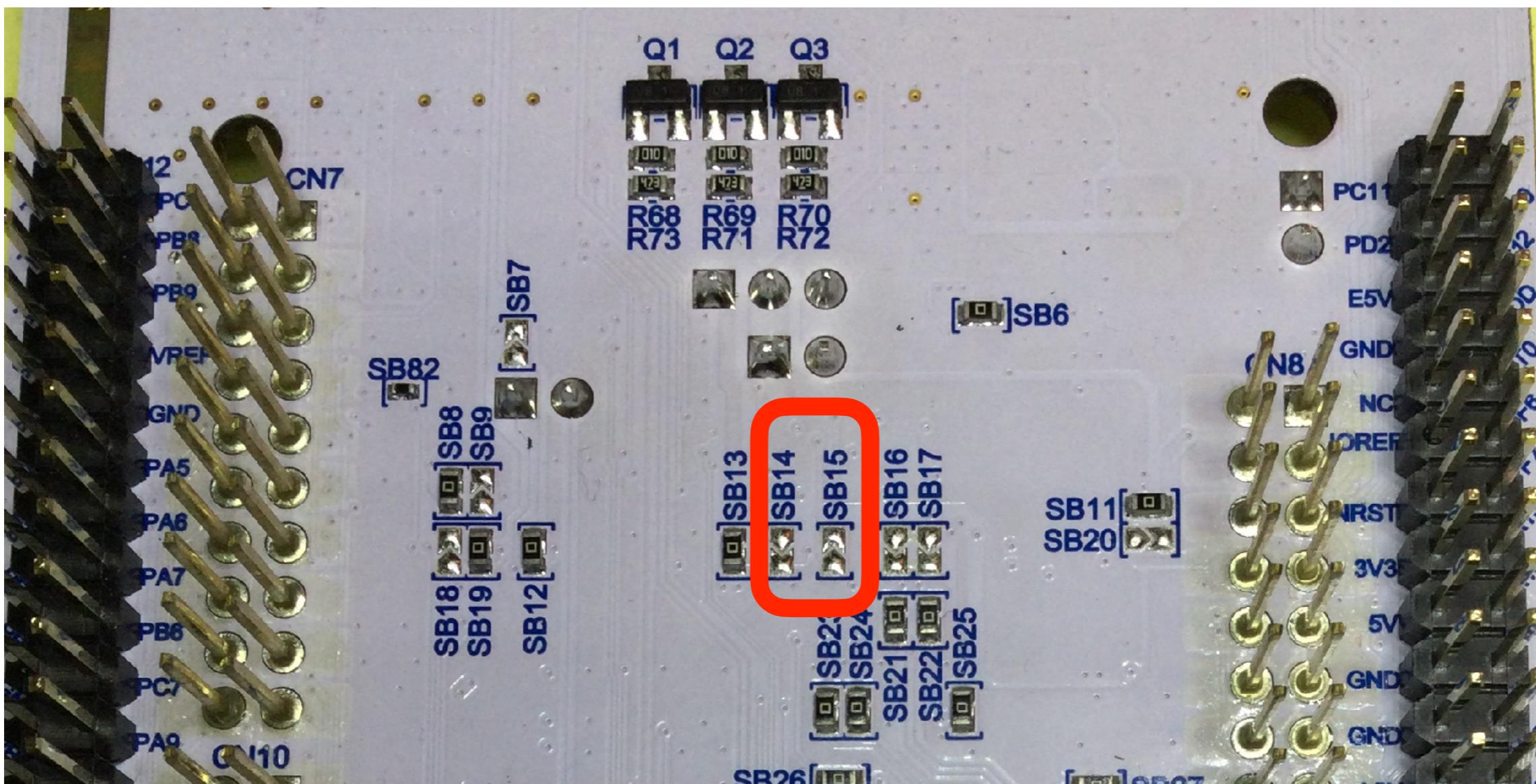
Schéma de connexion

Attention ! ne pas se tromper
dans la connexion du +24V, il n'y
a pas de diode de protection.



Avertissement (1/2)

Pour que la rotation de l'encodeur soit pris en compte, il faut effectuer deux ponts de soudure marqués **SB14** et **SB15** au dos de la carte **NUCLEO-H743ZI2**.



Avertissement (2/2)

STM32Duino version 1.9.0 présente deux bugs, qu'il faut corriger.

La lecture de l'entrée analogique EA0 renvoie toujours 0. L'entrée EA0 est connectée sur PA6, et STM32Duino v1.9.0 ne reconnaît pas PA6 comme une entrée analogique.

Il faut modifier `~/Library/Arduino15/packages/STM32/hardware/stm32/1.9.0/variants/NUCLEO_H743ZI/PeripheralPins.c`, en décommentant la ligne 57 :

```
{PA_6, ADC1, STM_PIN_DATA_EXT(STM_MODE_ANALOG, GPIO_NOPULL, 0, 3, 0)}, // ADC1_INP3
```

Ce bug a été signalé : https://github.com/stm32duino/Arduino_Core_STM32/issues/1277

La sortie TOR7 n'est jamais active. La sortie TOR7 est commandée par PG9, et STM32Duino v1.9.0 ne permet pas de programmer ce port comme une sortie logique.

Il faut modifier `~/Library/Arduino15/packages/STM32/hardware/stm32/1.9.0/variants/NUCLEO_H743ZI/variant.h`, en changeant la ligne 174 :

```
#define NUM_DIGITAL_PINS 99
```

En :

```
#define NUM_DIGITAL_PINS 100
```

Ce bug a été signalé : https://github.com/stm32duino/Arduino_Core_STM32/issues/1276

Interfaces utilisateur

Afficheur LCD

L'afficheur LCD contient 4 lignes de 20 caractères.

Sa gestion est effectuée par la librairie **LiquidCrystal** (<https://www.arduino.cc/en/Reference/LiquidCrystal>).

Son initialisation est effectuée par la fonction `configurerCarteH743LHEEA` (définie dans `STM32H743-configuration-lheea.cpp`), à appeler dans `setup`.

La variable à utiliser est `lcd` (déclarée dans `STM32H743-configuration-lheea.h`).

En résumé :

- `lcd.setCursor (x, y)` déplace la curseur à la ligne x ($0 \leq x \leq 3$), colonne y ($0 \leq y \leq 19$) ;
- `lcd.print (v)` imprime v à l'emplacement du curseur ; celui-ci est avancé du nombre de caractères écrits ;
- attention, ne pas déborder d'une ligne, il n'y a pas prolongement à la ligne suivante : par exemple, la suite de la 1^{re} ligne est la 3^e.

Leds

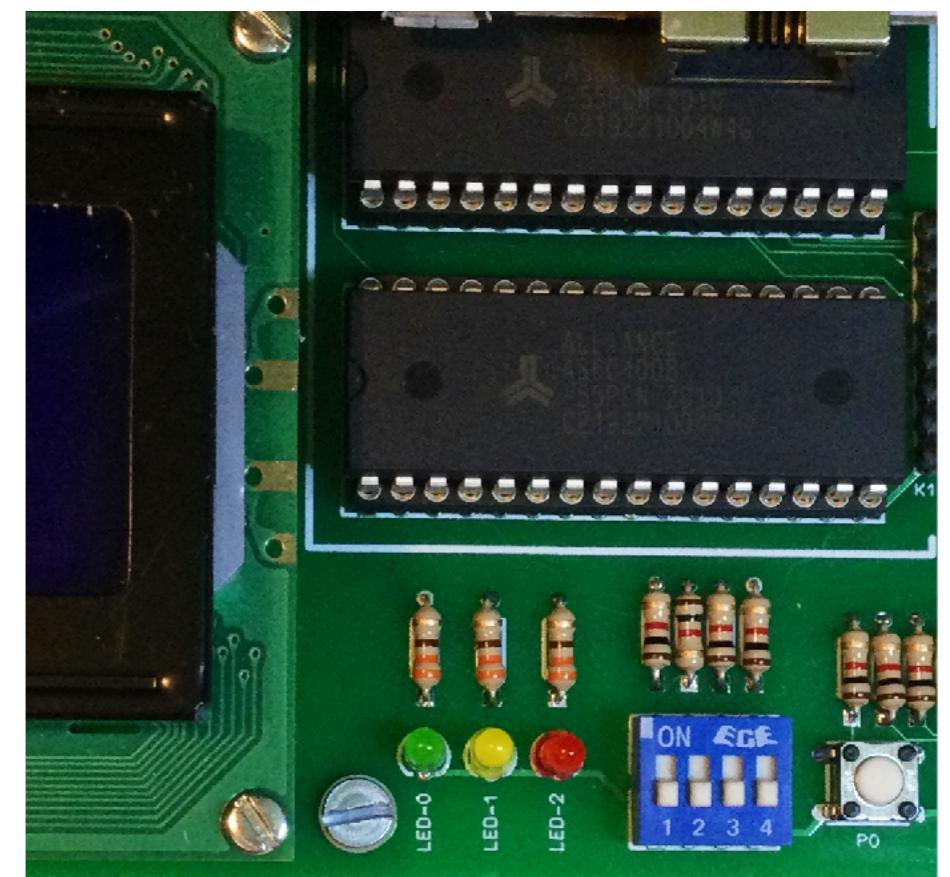
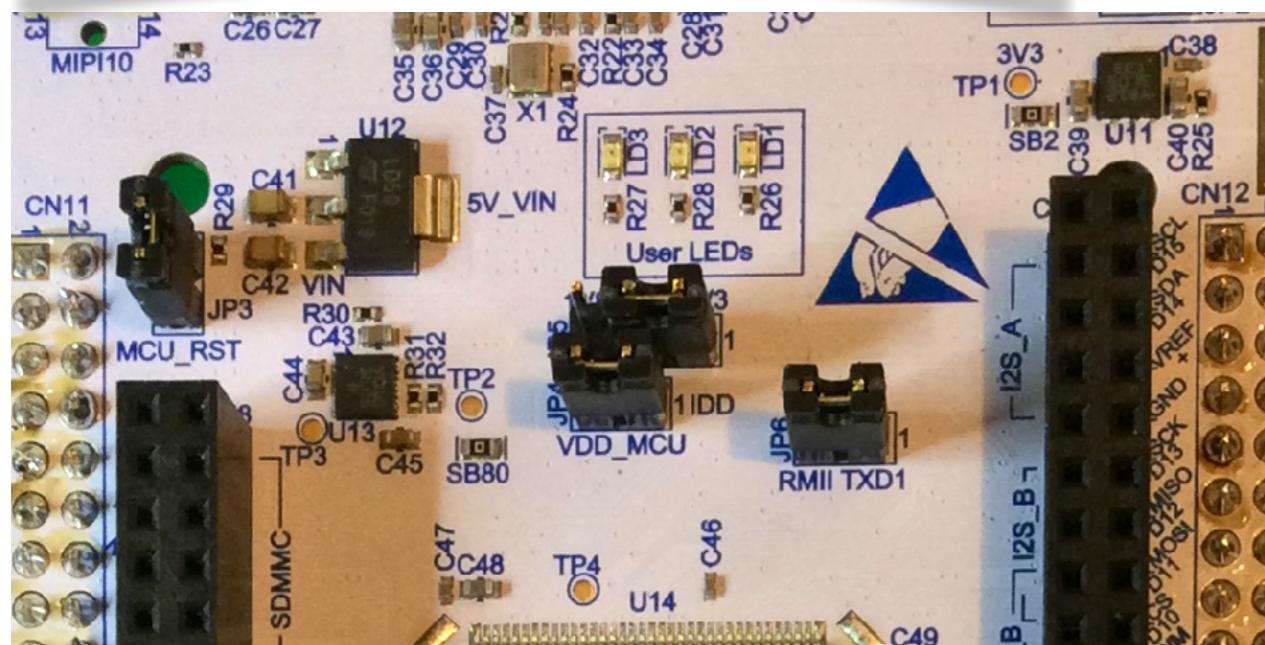
Il y a 6 leds disponibles :

- 3 leds sur la carte Nucleo ;
- 3 leds sur la carte LHEEA.

À chacune correspond un nom (déclaré dans STM32H743-configuration-1heea.h), ce qui permet de les identifier plus facilement. La fonction configurerCarteH743LHEEA (définie dans STM32H743-configuration-1heea.cpp, à appeler dans setup) programme les ports correspondants en sortie.

Écrire un niveau haut par digitalWrite (led, HIGH) allume la led, écrire un niveau bas par digitalWrite (led, LOW) l'éteint.

```
static const uint8_t LED_0_Verte = PA0 ;
static const uint8_t LED_1_Jaune = PA3 ;
static const uint8_t LED_2_Rouge = PB2 ;
static const uint8_t NUCLEO_LD1_Verte = PB0 ;
static const uint8_t NUCLEO_LD2_Jaune = PE1 ;
static const uint8_t NUCLEO_LD3_Rouge = PB14 ;
```



Interrupteurs DIL

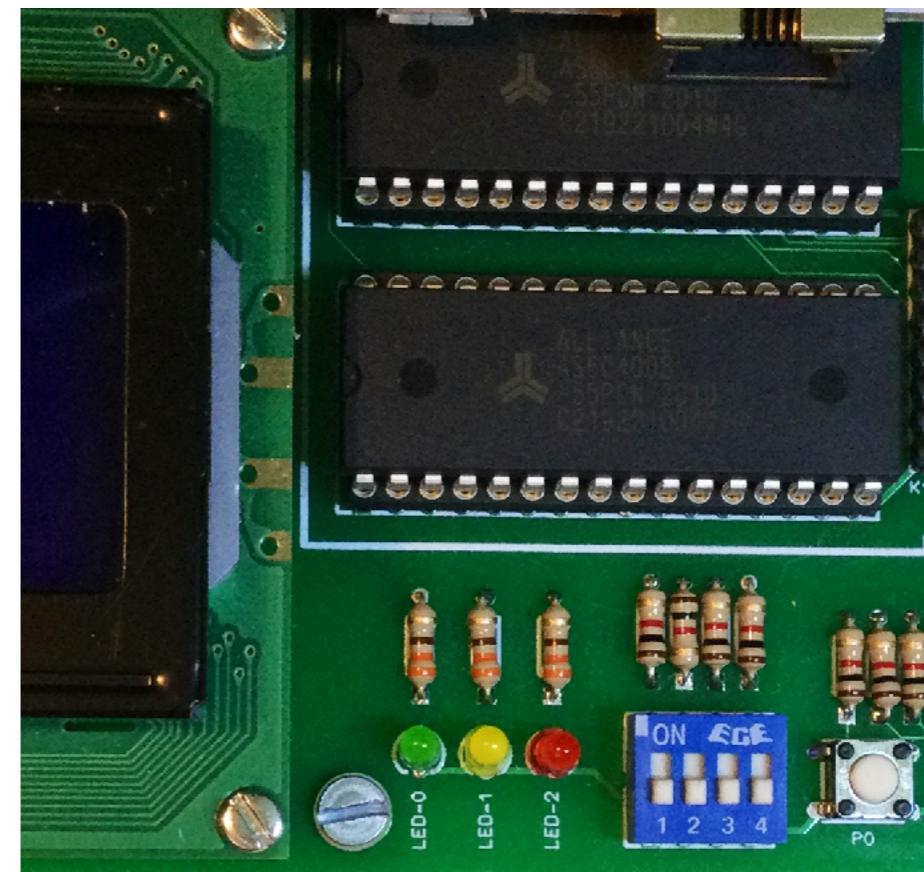
Il y a 4 interrupteurs DIL sur la carte LHEEA.

À chacun correspond un nom (déclaré dans STM32H743-configuration-lheea.h), ce qui permet de les identifier plus facilement. La fonction configurerCarteH743LHEEA (définie dans STM32H743-configuration-lheea.cpp, à appeler dans setup) programme les ports correspondants en entrée (*pullup* activé).

```
static const uint8_t INTER_DIL_1 = PG14 ;  
static const uint8_t INTER_DIL_2 = PB11 ;  
static const uint8_t INTER_DIL_3 = PB9 ;  
static const uint8_t INTER_DIL_4 = PB8 ;
```

Attention, la fonction digitalRead(*interDIL*) renvoie :

- HIGH si l'interrupteur est à OFF ;
- LOW si l'interrupteur est à ON.



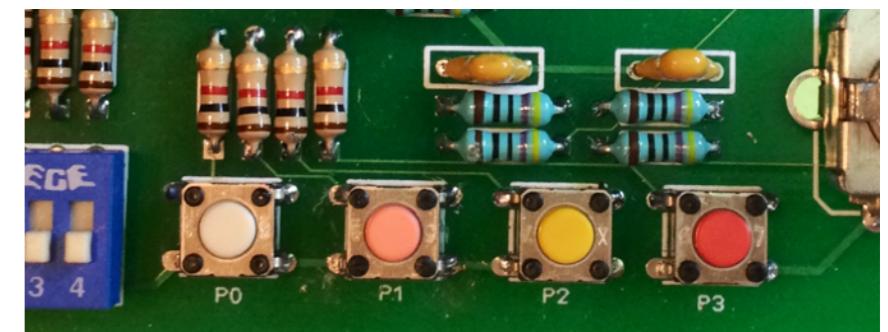
Poussoirs

Il y a 5 poussoirs :

- un poussoir (bleu) sur la carte Nucleo ;
- quatre poussoirs (blanc, rose, jaune, rouge) sur la carte LHEEA.

À chacun correspond un nom (déclaré dans STM32H743-configuration-lheea.h), ce qui permet de les identifier plus facilement. La fonction configurerCarteH743LHEEA (définie dans STM32H743-configuration-lheea.cpp, à appeler dans setup) programme les ports correspondants en entrée.

```
static const uint8_t POUSSOIR_P0_BLANC = PE0 ;
static const uint8_t POUSSOIR_P1_ROSE = PE2 ;
static const uint8_t POUSSOIR_P2_JAUNE = PE5 ;
static const uint8_t POUSSOIR_P3_ROUGE = PE6 ;
static const uint8_t POUSSOIR_NUCLEO_BLEU = PC13 ;
```



La fonction `digitalRead(POUSSOIR_NUCLEO_BLEU)` renvoie :

- LOW si le poussoir est relâché ;
- HIGH si le poussoir est appuyé.

Pour les autres poussoirs, c'est l'inverse, la fonction `digitalRead(poussoir)` renvoie :

- HIGH si le poussoir est relâché ;
- LOW si le poussoir est appuyé.

Encodeur (1/2)

L'encodeur a deux fonctions :

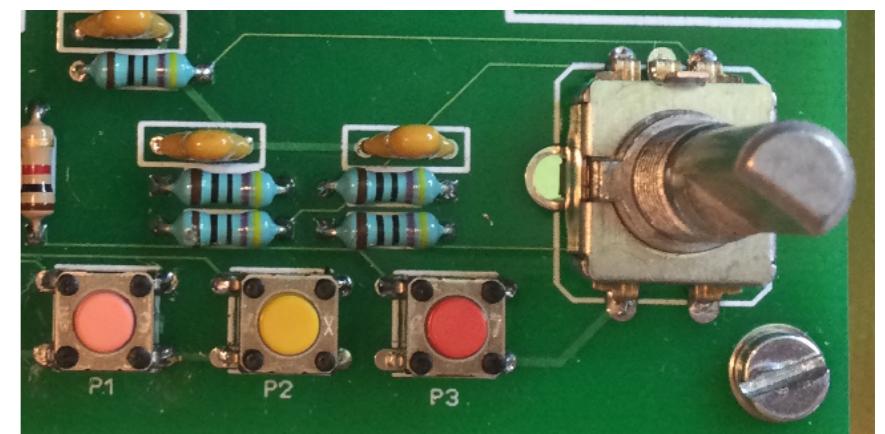
- la première, similaire à celle d'un poussoir ;
- la seconde est la rotation.

Poussoir. Pour la fonction poussoir, utiliser la constante ENCODEUR_CLIC déclarée dans STM32H743-configuration-lheea.h. La fonction configurerCarteH743LHEEA (définie dans STM32H743-configuration-lheea.cpp, à appeler dans setup) programme le port correspondant en entrée.

```
static const uint8_t ENCODEUR_CLIC = PC6 ;
```

La fonction digitalRead(*ENCODEUR_CLIC*) renvoie :

- HIGH si le poussoir est relâché ;
- LOW si le poussoir est appuyé.



Rotation. Prendre en charge la rotation de l'encodeur se fait en deux étapes :

- dans la fonction setup, appeler fixerGammeEncodeur pour fixer la plage des valeurs ;
- ensuite, appeler quand on veut valeurEncodeur pour récupérer la valeur de l'encodeur.

Encodeur (2/2)

La fonction `fixerGammeEncodeur` a deux arguments :

`fixerGammeEncodeur (borneInf, borneSup)`

où :

- `borneInf` est la borne inférieure de la plage des valeurs ;
- `borneSup` est la borne supérieure de la plage des valeurs.

Des valeur négatives sont acceptées (les arguments formels sont de type `int32_t`). Il faut appeler cette fonction avec `borneInf ≤ borneSup`.

Si `borneInf == borneSup`, la fonction `valeurEncodeur` renvoie toujours la valeur commune, indépendamment de la rotation de l'encodeur.

Si `borneInf < borneSup`, la fonction `valeurEncodeur` renvoie toujours une valeur entière dans l'intervalle `[borneInf, borneSup]` :

- une rotation dans le sens trigonométrique décrémente la valeur renvoyée, jusqu'à saturer à `borneInf` ;
- une rotation dans le sens des aiguilles d'une montre incrémente la valeur renvoyée, jusqu'à saturer à `borneSup`.

Initialement, par défaut, `borneInf == borneSup == 0`. La fonction `valeurEncodeur` renvoie alors toujours 0.

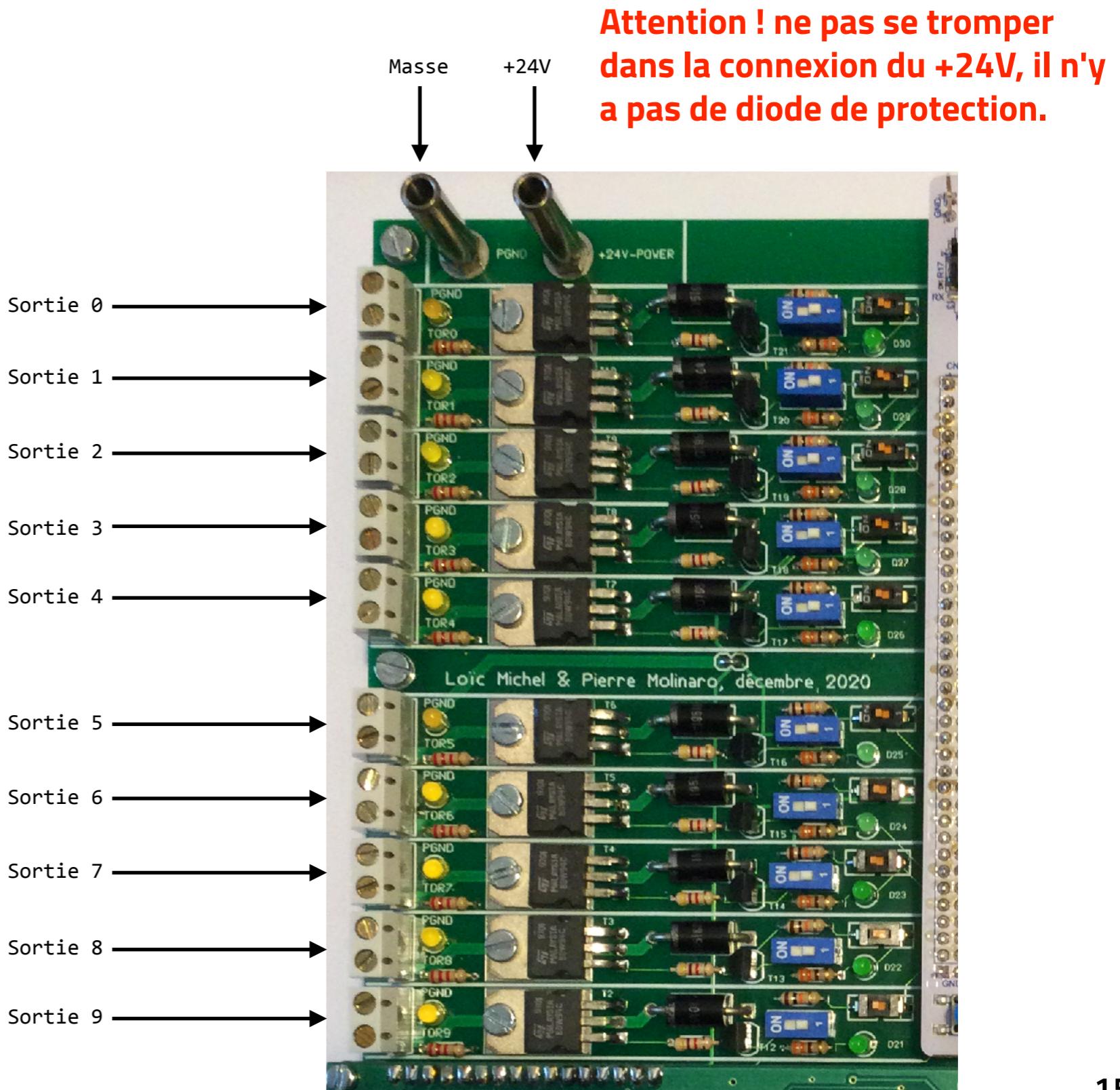
Sorties TOR

Les 10 sorties TOR

La carte contient 10 sorties de puissances, appelées sorties TOR (*Tout-Ou-Rien*).

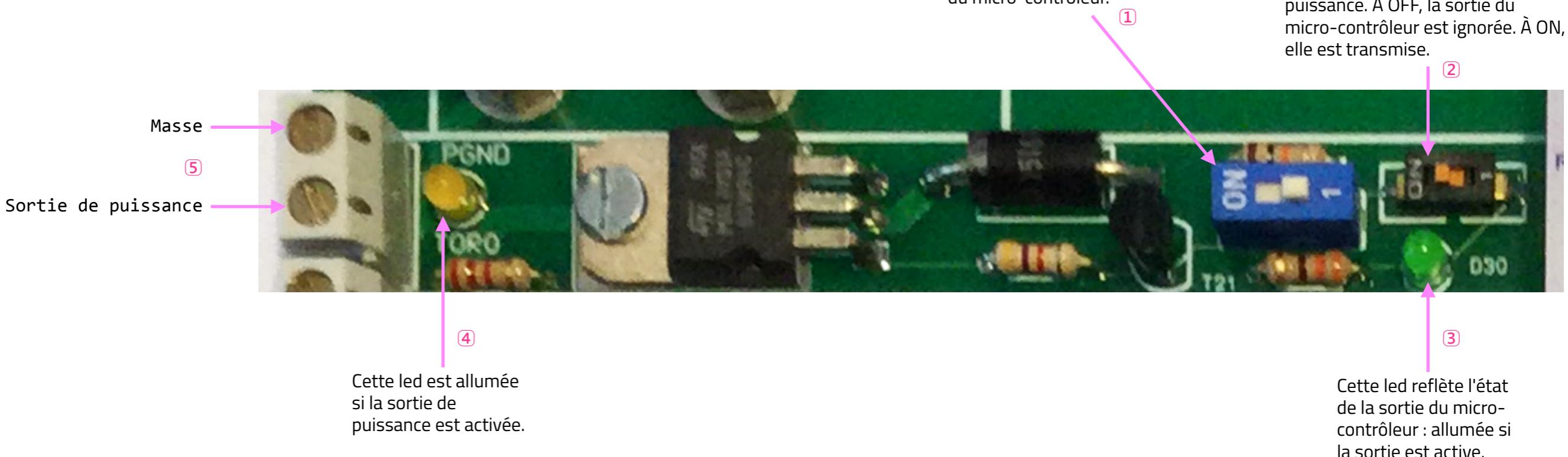
Pour la connexion de l'alimentation de puissance, utiliser exclusivement les bornes PGND et +24V.

La carte présente d'autres bornes de masse, mais il est très déconseillé de les utiliser pour véhiculer les courants importants des sorties de puissance.



Détails d'une sortie TOR

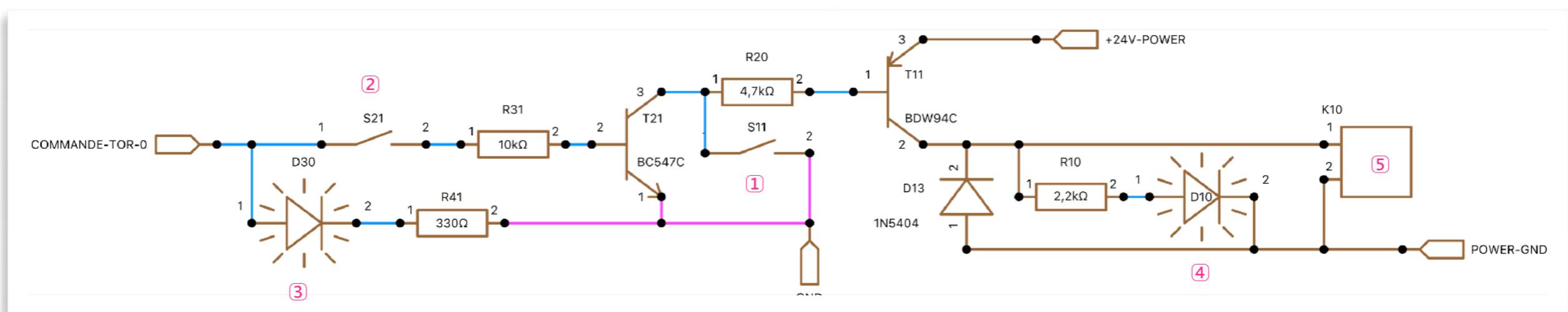
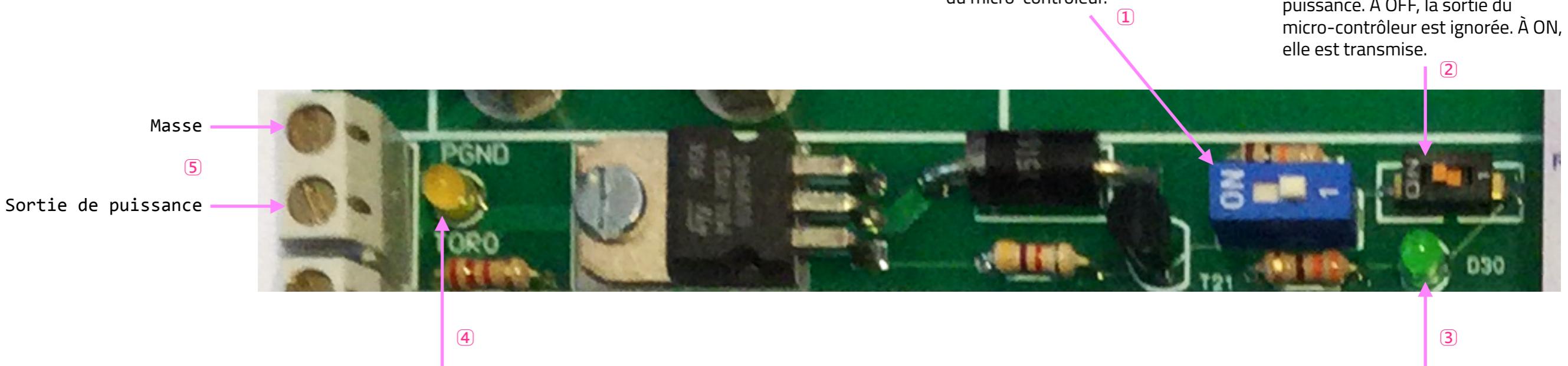
Les 10 sorties sont disposées de manière identiques.



Interrupteur ①	Interrupteur ②	Led ③	Led ④	Sortie de puissance ⑤
ON	X	X	Allumée	Active
OFF	OFF	X	Éteinte	Inactive
OFF	ON	Éteinte	Éteinte	Inactive
OFF	ON	Allumée	Allumée	Active

Schéma d'une sortie TOR

Les 10 sorties sont disposées de manière identiques.



Commande d'une sortie TOR

L'initialisation des ports correspondants aux sorties TOR est effectuée par la fonction configurerCarteH743LHEEA (définie dans STM32H743-configuration-1heea.cpp), à appeler dans setup. L'initialisation place les sorties du micro-contrôleur dans l'état inactif (la led ③ éteinte).

Pour commander une sortie, la fonction à utiliser est activerSortieTOR (déclarée dans STM32H743-configuration-1heea.h) :

```
void activerSortieTOR (const uint32_t inIndex, const bool inValue) ;
```

Les arguments sont :

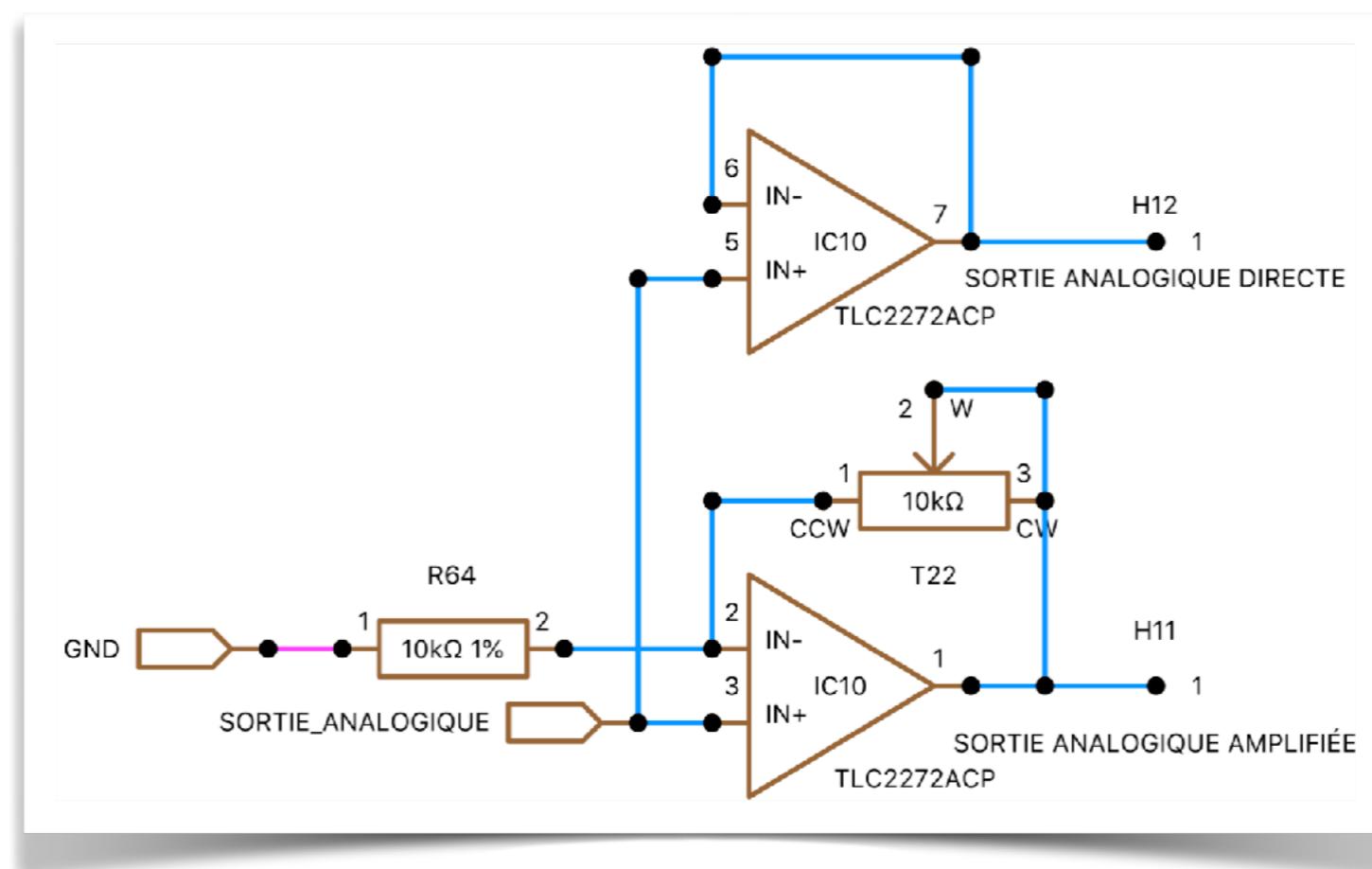
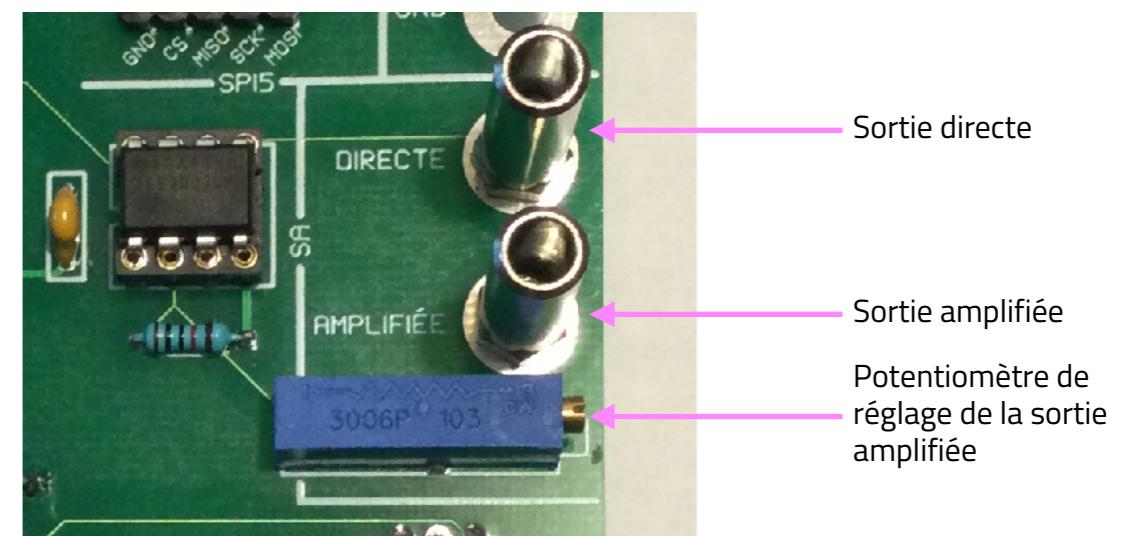
- **inIndex** : le numéro de la sortie (entre 0 et 9) ; si supérieur à 9, l'appel n'a pas d'effet ;
- **inValue** : à true active la sortie du micro-contrôleur (la led ③ est allumée), à false inactive la sortie du micro-contrôleur (la led ③ est éteinte).

Sortie analogique

Sortie analogique

Le micro-contrôleur intègre un convertisseur numérique / analogique (port : PA4) qui conduit deux sorties à travers deux amplificateur opérationnels :

- la sortie DIRECTE, dont la plage de tension est [0, 3,3V] ;
- la sortie AMPLIFIÉE qui est une image amplifiée de la sortie, réglable grâce au potentiomètre.



Commander la sortie analogique

```
void commanderSortieAnalogique (const uint8_t inValue) ;
```

La fonction `commanderSortieAnalogique` a un argument : `inValue`, un entier non signé sur 8 bits qui code la valeur analogique :

- `inValue = 0` -> la sortie directe est à 0V ;
- `inValue = 255` -> la sortie directe est à 3,3V.

La tension de la sortie directe est proportionnelle à `inValue`, par exemple, pour la valeur 100, la tension de la sortie directe est $100 * 3,3V / 255 = 1,29V$.

La tension de la sortie amplifiée est aussi proportionnelle à `inValue`, le facteur de proportionnalité dépend du réglage du potentiomètre :

- tourné au maximum dans le sens des aiguilles d'une montre, la tension de la sortie amplifiée est la même que celle de la sortie directe (l'amplificateur opérationnel est configuré en *suiveur*) ;
- tourné au maximum dans le sens trigonométrique, la tension de la sortie amplifiée est le double que celle de la sortie directe (l'amplificateur opérationnel est configuré en *amplificateur non inverseur*) ; en réalité, la tension est un peu inférieure au double, la résistance du potentiomètre a une précision de 10%.

Entrées analogiques

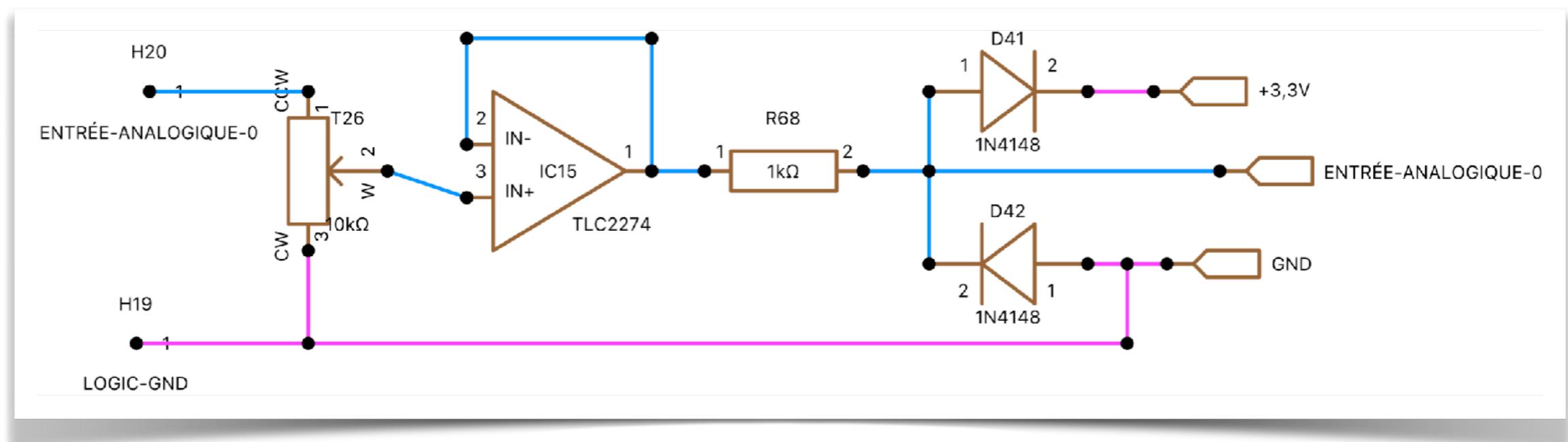
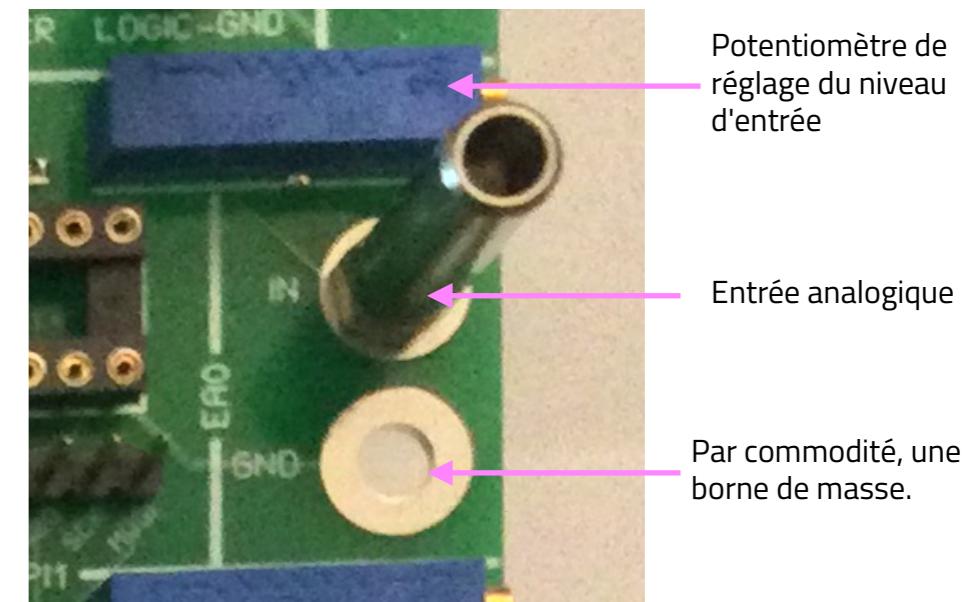
Les entrées analogiques

La carte contient 4 entrées analogiques dont la disposition et la configuration sont identiques.

Le potentiomètre permet de régler le niveau d'entrée :

- tourné au maximum dans le sens des aiguilles d'une montre, la tension de l'entrée analogique du micro-contrôleur sera toujours à 0 ;
- tourné au maximum dans le sens trigonométrique, la tension de l'entrée analogique du micro-contrôleur sera celle de la borne d'entrée.

Deux diodes 1N4148 protègent le micro-contrôleur d'une surtension à sur l'entrée analogique.



Lire une entrée analogique

```
uint16_t lireEntreeAnalogique (const uint32_t inNumeroEntree) ;
```

La fonction `lireEntreeAnalogique` a un argument, le numéro (0 à 3) de l'entrée que l'on veut lire (si l'argument est > 3, la valeur renvoyée est toujours 0), et retourne une image de l'entrée analogique.

L'acquisition s'effectue sur 12 bits, c'est-à-dire :

- entrée micro-contrôleur à 0V -> valeur renvoyée 0 ;
- entrée micro-contrôleur à 3,3V -> valeur renvoyée 4095.

La valeur renvoyée est proportionnelle la tension de l'entrée analogique ; par exemple, pour une tension 1,29V, la valeur renvoyée est $1,29V * 4095 / 3,3V = 1600$.

RAM externe

Plan de la carte (1/16)

FLASHs et EEPROMs externes

Plan de la carte (1/16)

Croquis d'exemple

Les croquis d'exemple

Les croquis d'exemple sont dans le répertoire `croquis-exemple-stm32duino` :

- `01-croquis-test-es-logiques` ;
- `02-croquis-test-es-analogiques` ;
- `03-croquis-test-sorties-tor` ;
- `04-croquis-exemple-fmc`.

Note. pensez à effectuer les opérations précédemment décrites :

- ponts de soudure marqués **SB14** et **SB15** au dos de la carte **NUCLEO-H743ZI2** ([voir à cette page](#)) ;
- modifier STM32Duino pour que PA6 soit reconnue comme une entrée analogique ([voir à cette page](#)) ;
- modifier STM32Duino pour que PG9 soit reconnue comme une sortie logique ([voir à cette page](#)).

Composition des croquis

Tout le code de gestion de la carte `croquis-exemple-stm32duino` est contenu dans cinq fichiers :

- `STM32H743-configuration-lheea.cpp`
- `STM32H743-configuration-lheea.h`
- `STM32H743-control-registers.h`
- `STM32H743-pin-modes.cpp`
- `STM32H743-pin-modes.h`

Pour les utiliser dans un croquis, il y a 2 possibilités :

- ① Les inclure simplement dans le répertoire du croquis, à côté du fichier `.ino`.
- ② Inclure les *liens symboliques* vers ces fichiers. C'est ce qui est fait pour tous les croquis d'exemple présentés dans la suite. Pour cela, **ne pas utiliser** *Créer un alias* du menu *Fichier* du Finder. Il faut exécuter dans le terminal :

```
cd répertoire_du_croquis
ln -s chemin/*.cpp .
ln -s chemin/*.h .
```

Attention, cette technique ne fonctionne pas sous Windows (Windows ne connaît pas les liens symboliques).

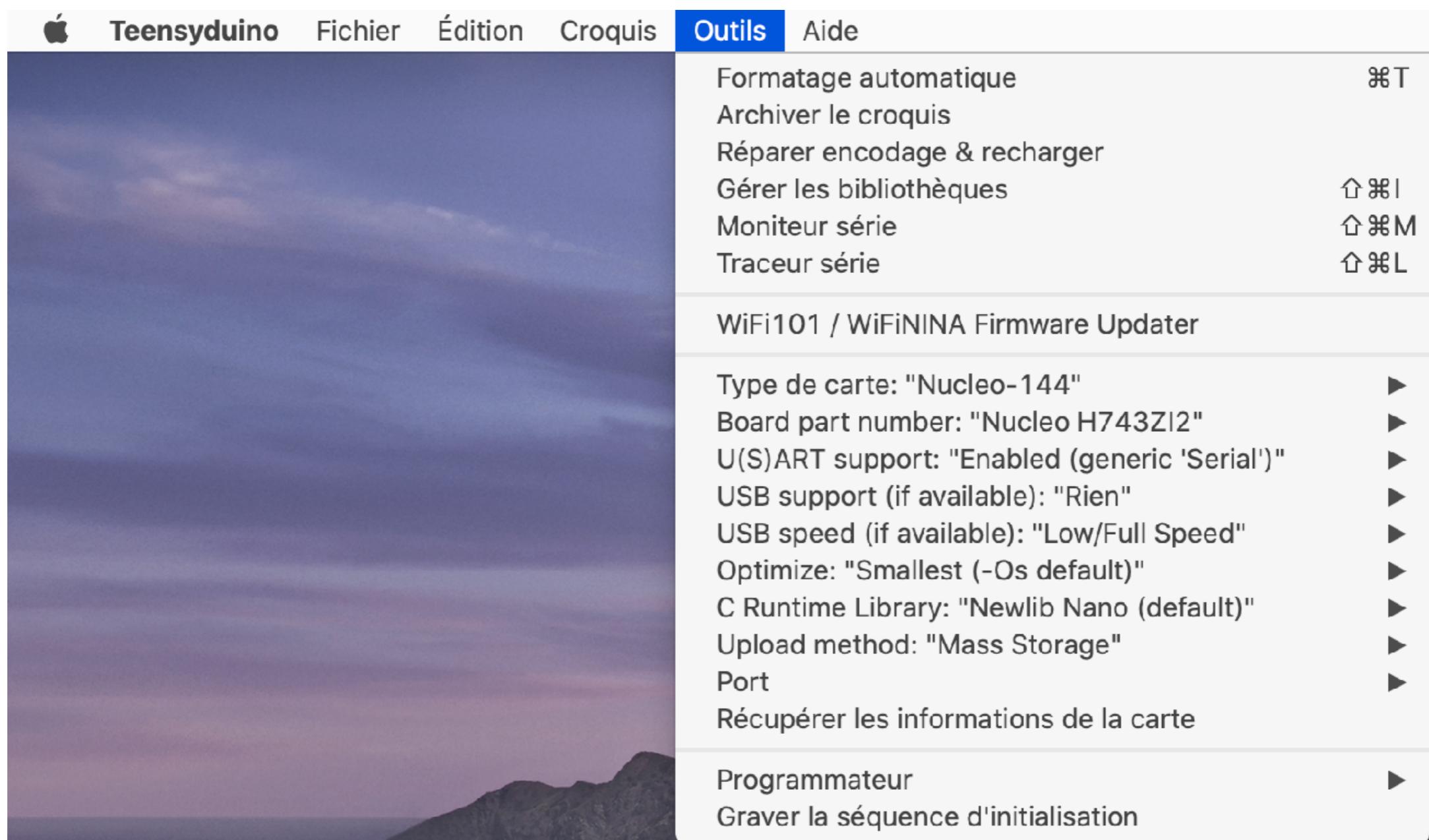
Dans la fonction **setup** du croquis, appeler la fonction **configurerCarteH743LHEEA**.

Sélection de la carte

La sélection de la carte doit être :

Type de carte : **Nucleo-144** ;

Board part number : **Nucleo H743ZI2** (**Attention, ne pas sélectionner Nucleo H743ZI, c'est une autre carte**).



Écriture d'un croquis

Pour écrire un croquis pour la carte asservissement, il faut suivre les règles suivantes dans le fichier `.ino` :

- ① Inclure `STM32H743-configuration-lheea.h` au début du fichier :

```
#include "STM32H743-configuration-lheea.h"
```

- ② Dans la fonction `setup`, appeler une et une seule fois la fonction `configurerCarteH743LHEEA` :

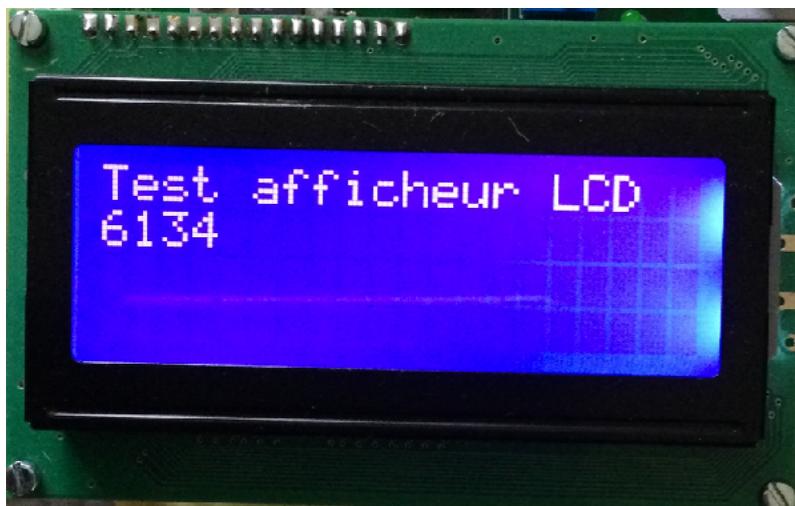
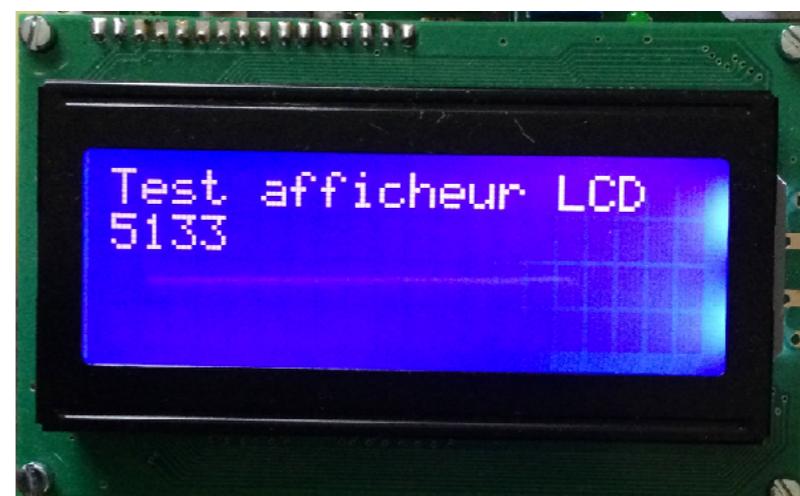
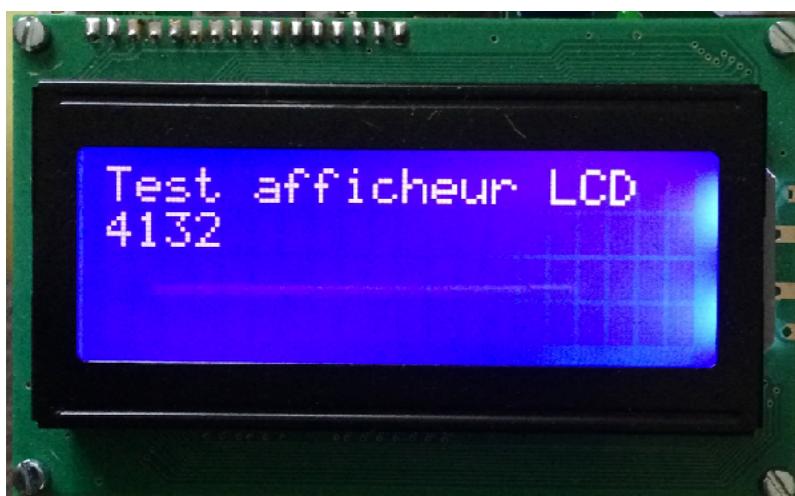
```
void setup () {  
    configurerCarteH743LHEEA () ;  
    ...  
}
```

- ③ Ensuite, toutes les fonctions citées dans `STM32H743-configuration-lheea.h` peuvent être appelées. L'objet des croquis d'exemple qui suivent est d'illustrer leur utilisation.

Croquis 01-afficheur-lcd

Ce croquis illustre l'utilisation de l'afficheur LCD.

Il montre aussi que la fonction **delay** ne permet pas d'avoir des exécutions vraiment périodiques (le croquis suivant va y remédier) : alors que le délai est 1000, la période est supérieure à cette valeur.



Croquis 02-ne-pas-utiliser-delay

Utiliser `delay` est simple, mais cela pose beaucoup de problèmes que l'on ne peut pas résoudre facilement lorsque l'on a besoin de gérer des actions avec des périodes différentes.

Le plus simple, pour chaque action périodique, est de maintenir une variable (`gInstantClignotementXXXX` et `gInstantAffichage` dans ce croquis d'exemple) qui détient la date de la prochaine exécution de l'action.

De cette façon :

- si la date d'exécution n'est pas atteinte, le processeur n'est pas immobilisé ;
- on peut facilement fixer indépendamment la période de chaque action.

Le croquis définit 4 actions périodiques :

- clignotement de la led verte de la carte Nucleo ;
- clignotement de la led jaune de la carte Nucleo ;
- clignotement de la led rouge de la carte Nucleo ;
- affichage de la date courante (noter que la dérive observée avec le croquis précédent a disparu).

Note : `millis()` retourne la date courante (nombre de millisecondes depuis le démarrage) dans un `uint32_t` (entier non signé de 32 bits). Il y a débordement au bout de $2^{32}-1$ millisecondes, soit plus de 49 jours. Après débordement, le code ne fonctionne plus.

Croquis 02-ne-pas-utiliser-delay

Utiliser `delay` est simple, mais cela pose beaucoup de problèmes que l'on ne peut pas résoudre facilement lorsque l'on a besoin de gérer des actions avec des périodes différentes.

Le plus simple, pour chaque action périodique, est de maintenir une variable (`gInstantClignotementXXXX` et `gInstantAffichage` dans ce croquis d'exemple) qui détient la date de la prochaine exécution de l'action.

De cette façon :

- si la date d'exécution n'est pas atteinte, le processeur n'est pas immobilisé ;
- on peut facilement fixer indépendamment la période de chaque action.

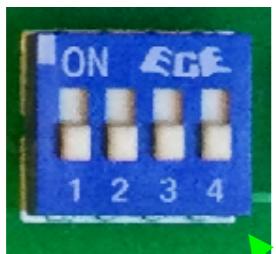
Le croquis définit 4 actions périodiques :

- clignotement de la led verte de la carte Nucleo ;
- clignotement de la led jaune de la carte Nucleo ;
- clignotement de la led rouge de la carte Nucleo ;
- affichage de la date courante (noter que la dérive observée avec le croquis précédent a disparu).

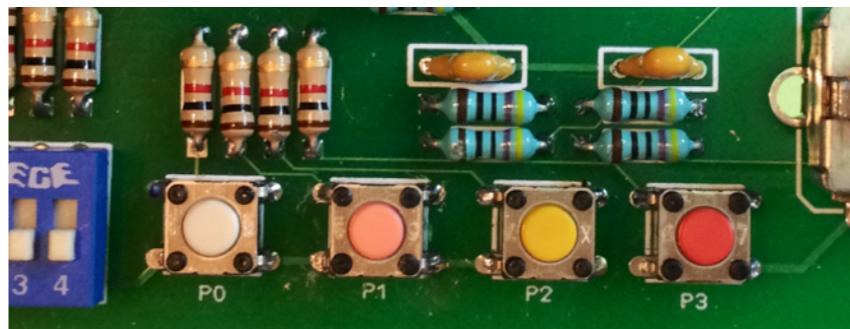
Note : `millis()` retourne la date courante (nombre de millisecondes depuis le démarrage) dans un `uint32_t` (entier non signé de 32 bits). Il y a débordement au bout de $2^{32}-1$ millisecondes, soit plus de 49 jours. Après débordement, le code ne fonctionne plus.

Croquis 03-croquis-test-es-logiques (1/2)

Ce croquis permet de tester les E/S logiques, et illustre l'utilisation des fonctions correspondantes.



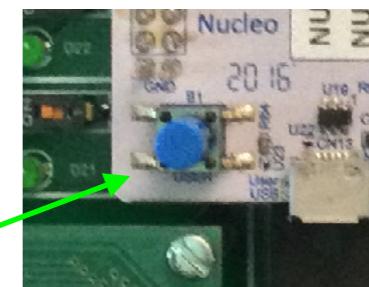
Interrupteur DIL : OFF -> 1, ON -> 0



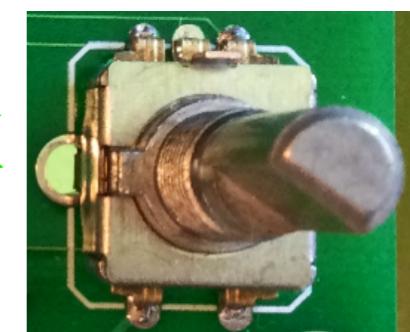
Poussoirs : relâché -> 1, appuyé -> 0



Poussoir bleu carte NUCLEO : relâché -> 0, appuyé -> 1



Poussoir de l'encodeur : relâché -> 1, appuyé -> 0



Rotation de l'encodeur : La gamme de l'encodeur est fixée à [-10, 10], par un appel à fixerGammeEncodeur dans la fonction setup (voir page suivante).

Croquis 03-croquis-test-es-logiques (2/2)

Utilisation de l'encodeur. Utiliser l'encodeur se fait en deux étapes :

- dans la fonction `setup`, appeler `fixerGammeEncodeur` pour fixer la plage des valeurs ;
- ensuite, appeler quand on veut `valeurEncodeur` pour récupérer la valeur de l'encodeur.

La fonction `fixerGammeEncodeur` a deux arguments :

`fixerGammeEncodeur (borneInf, borneSup)`

où :

- *borneInf* est la borne inférieure de la plage des valeurs ;
- *borneSup* est la borne supérieure de la plage des valeurs.

Des valeur négatives sont acceptées (les arguments formels sont de type `int32_t`). Il faut appeler cette fonction avec $\text{borneInf} \leq \text{borneSup}$.

Si $\text{borneInf} == \text{borneSup}$, la fonction `valeurEncodeur` renvoie toujours la valeur commune, indépendamment de la rotation de l'encodeur.

Si $\text{borneInf} < \text{borneSup}$, la fonction `valeurEncodeur` renvoie toujours une valeur entière dans l'intervalle $[\text{borneInf}, \text{borneSup}]$:

- une rotation dans le sens trigonométrique décrémente la valeur renvoyée, jusqu'à saturer à *borneInf* ;
- une rotation dans le sens des aiguilles d'une montre incrémente la valeur renvoyée, jusqu'à saturer à *borneSup*.

Initialement, par défaut, $\text{borneInf} == \text{borneSup} == 0$. La fonction `valeurEncodeur` renvoie alors toujours 0.

Croquis 04-croquis-test-es-analogiques

Ce croquis teste la sortie analogique et les quatre entrées logiques :

- la gamme de l'encodeur est fixée à [0, 255] ;
- la valeur de l'encodeur est envoyée sur la sortie analogique ;
- toutes les secondes, les quatre entrées analogiques sont lues et affichées.



Le nombre après SA (ici 100) est directement la valeur issue de l'encodeur rotatif. Cette valeur est envoyée sur la sortie analogique. La valeur théorique de la sortie DIRECTE est affichée, elle vaut $100 * 3,3V / 255 = 1,29V$; le multi-mètre affiche une valeur voisine : 1,30V.

Les quatre entrées analogiques sont affichées sur les deux dernières lignes. Les acquisitions sont sur 12 bits, la gamme des valeurs est donc [0, 4095].

Dans l'affichage ci-contre, les entrées analogiques 0, 1 et 3 ne sont pas connectées : leur valeur théorique est 0.



L'entrée analogique 2 est connectée à la sortie analogique DIRECTE. Le potentiomètre de réglage de l'entrée analogique 2 est réglé de façon à réaliser un gain de 1 (il est tourné au maximum dans le sens trigonométrique). La résolution de la sortie analogique est de 8 bits, celle de l'entrée 12 bits, la valeur théorique qui devrait être affichée pour EA2 est 2^{12-8} fois la valeur de l'encodeur, soit 1600.

Croquis 05-croquis-test-sorties-tor

Ce croquis effectue indéfiniment l'activation pendant une seconde des sorties TOR, dans l'ordre
TOR0, TOR1, ... TOR9, TOR0, ...

Croquis 06-croquis-test-ram-externe

Ce croquis effectue chaque seconde un test exhaustif de l'accès à la RAM externe.

le code du test est la fonction `testRamExterne`. Celle-ci écrit une valeur particulière dans chaque octet de la RAM externe (dont la taille est $4 * 512 * 1024$ octets), puis relie cette valeur et la vérifie.

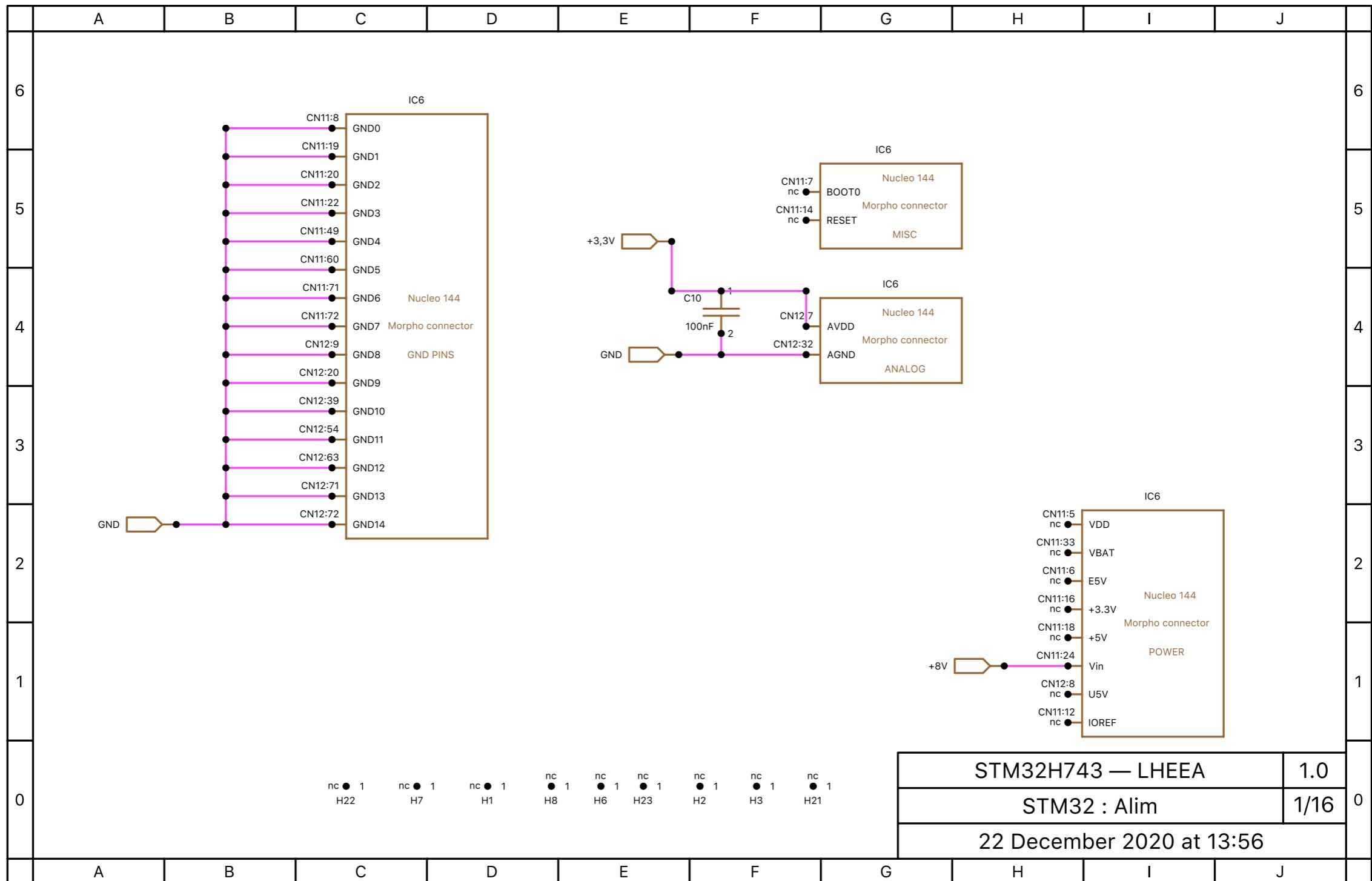
Le nombre cumulé d'erreurs est affiché à la dernière ligne. Il doit toujours être nul.

La durée d'un test est environ 419 ms. ceci permet de calculer le temps d'une lecture ou d'une écriture d'un octet en RAM externe : $419 \text{ ms} / (2 * 4 * 512 * 1024) \approx 100 \text{ ns}$.

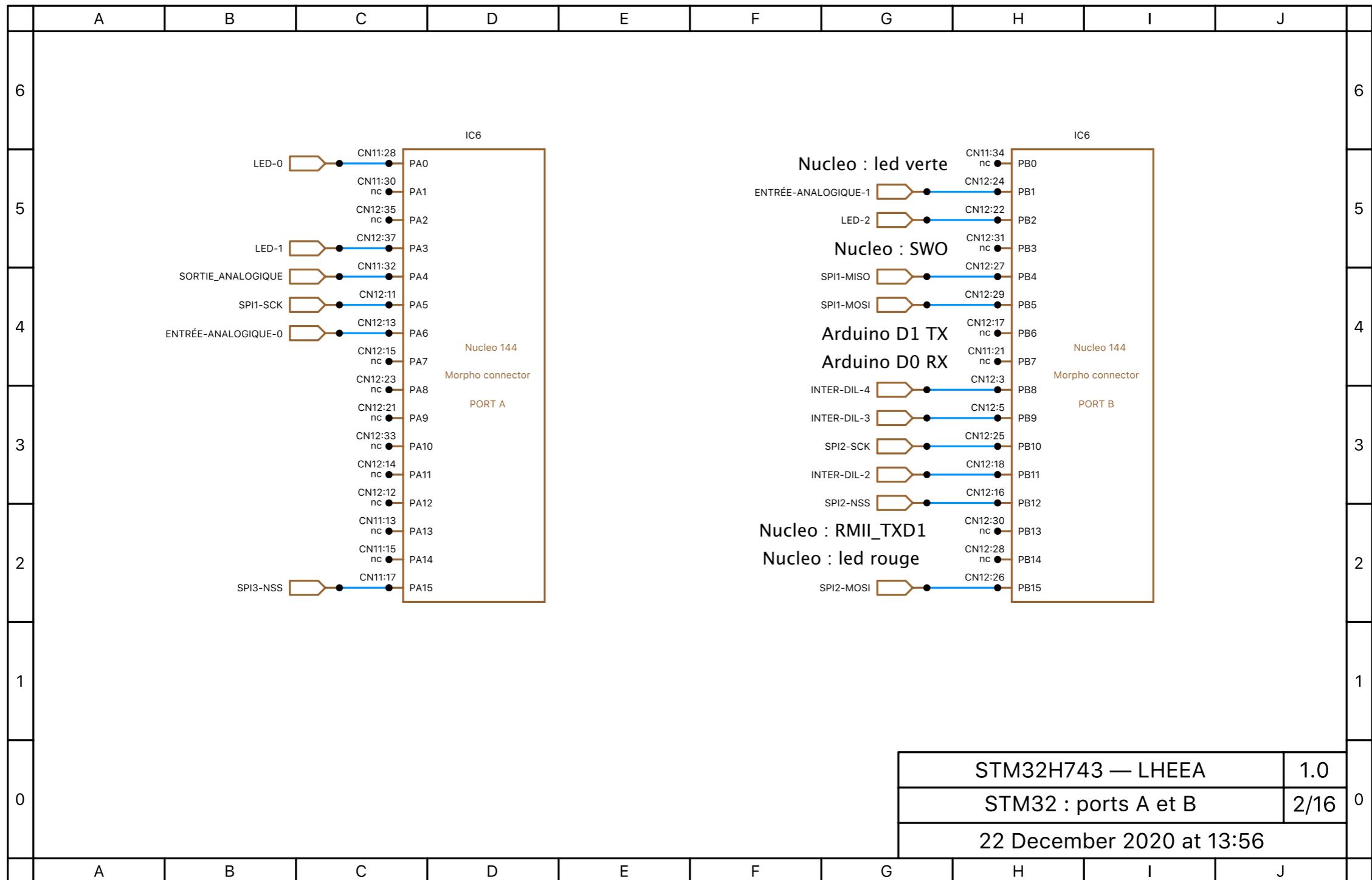


Plan de la carte

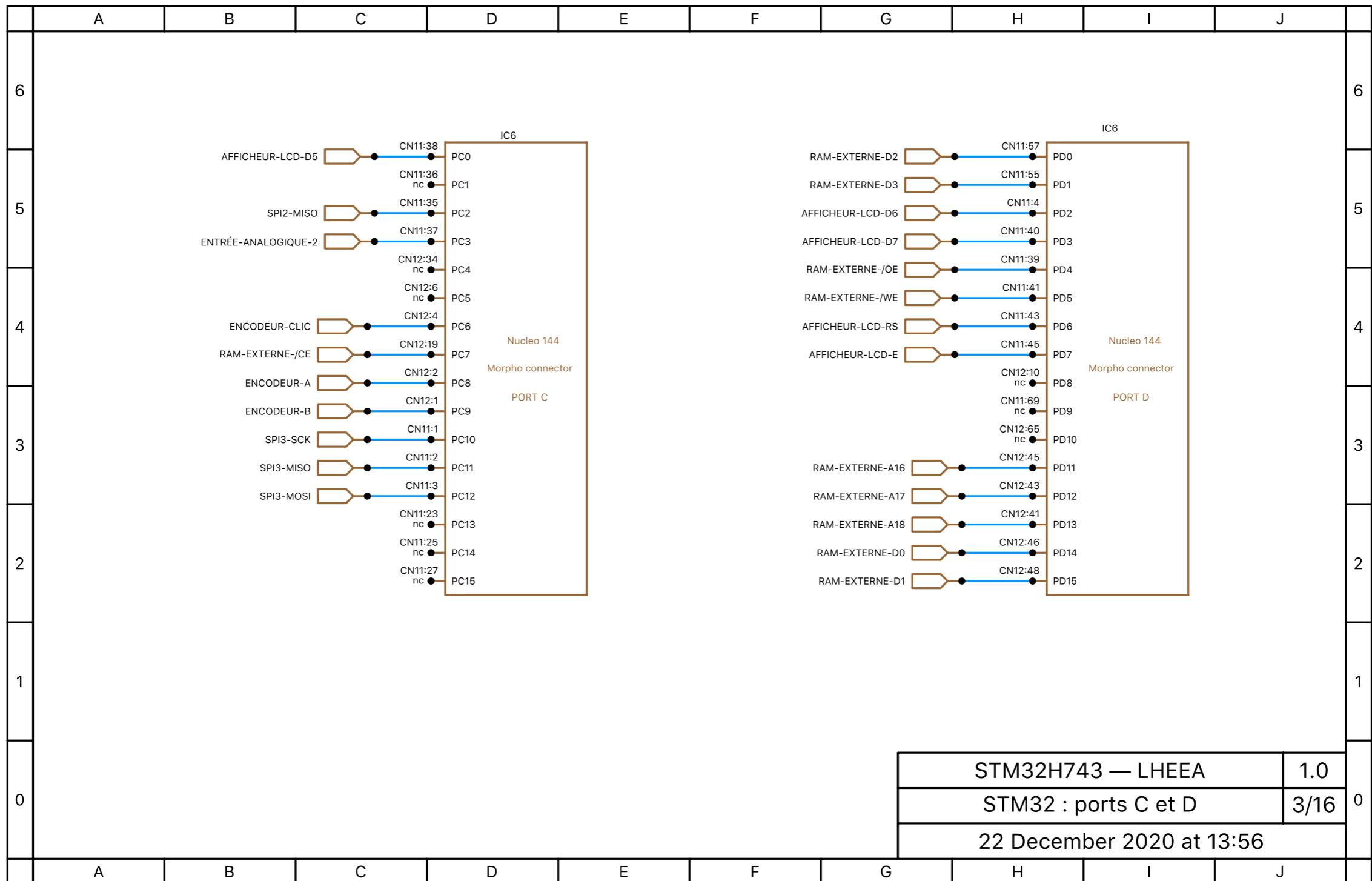
Plan de la carte (1/16)



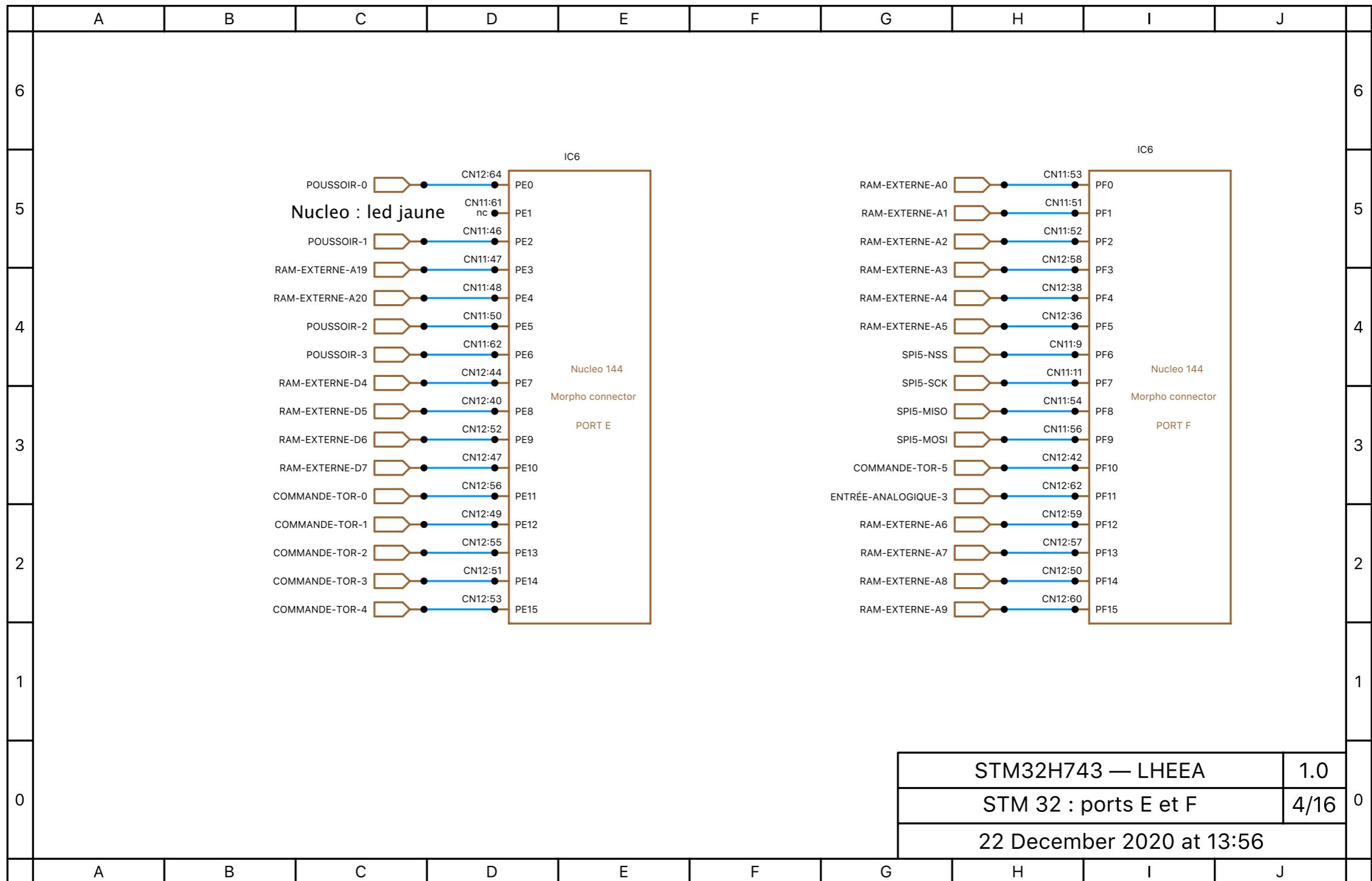
Plan de la carte (2/16)



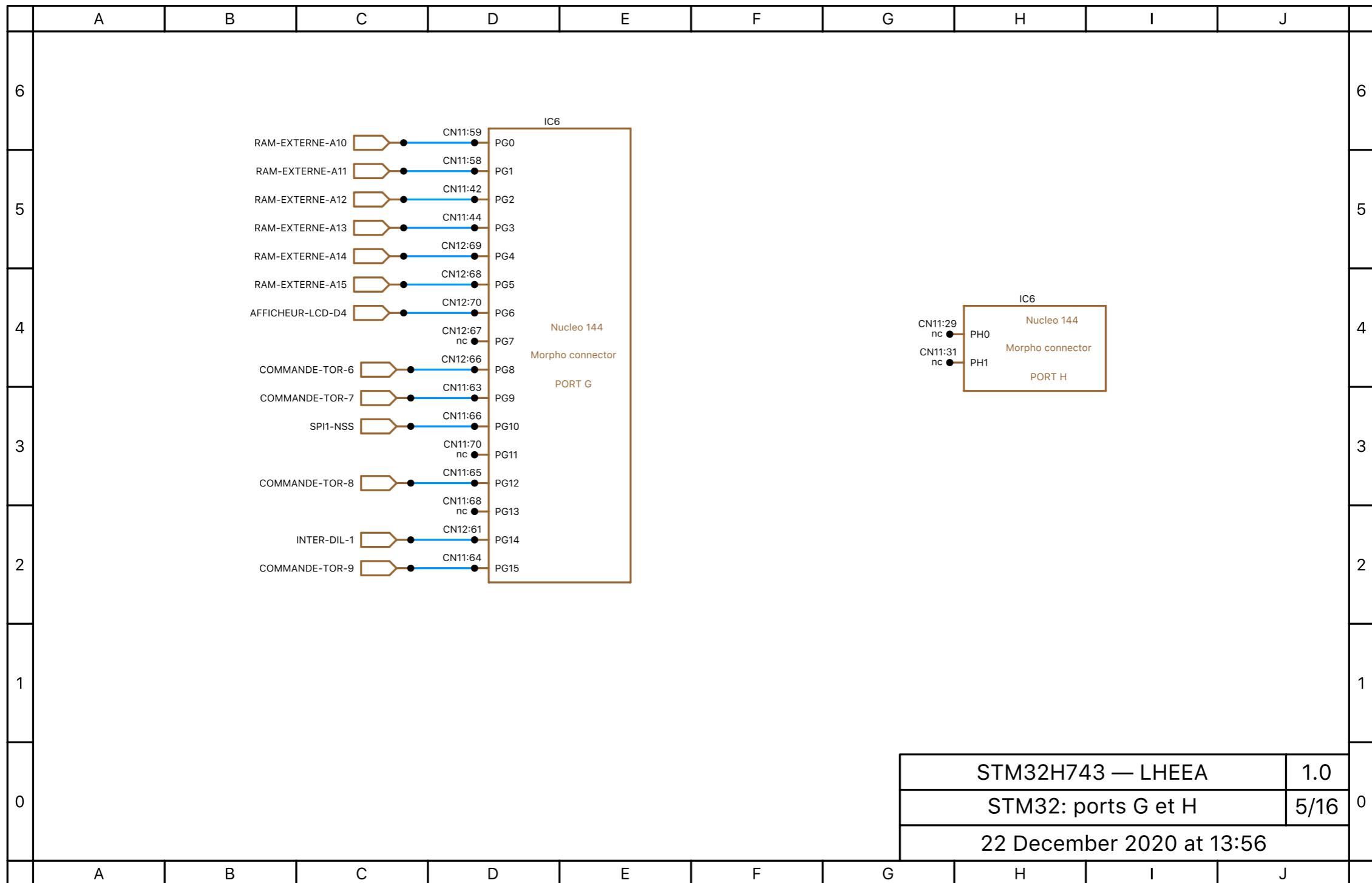
Plan de la carte (3/16)



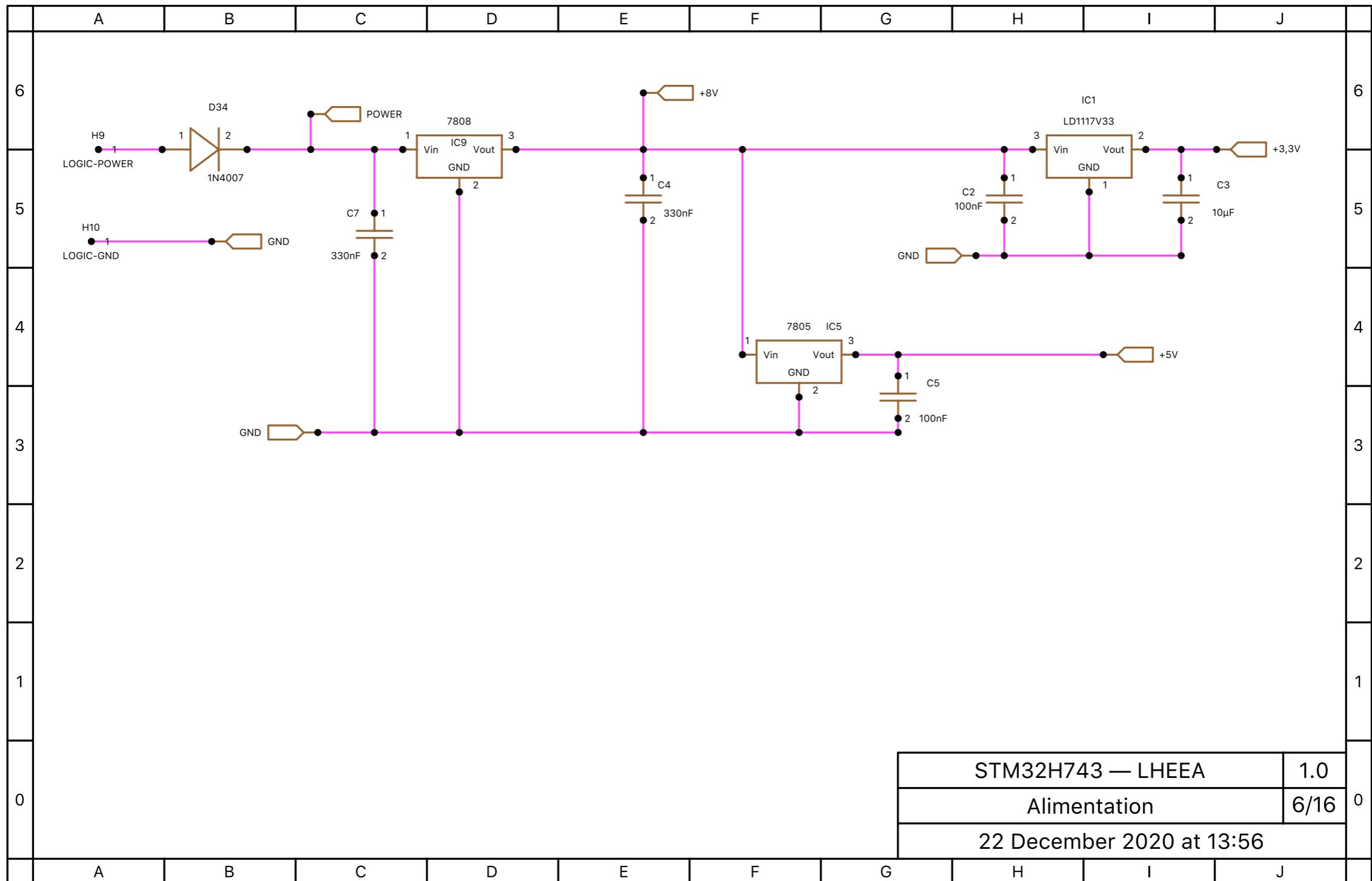
Plan de la carte (4/16)



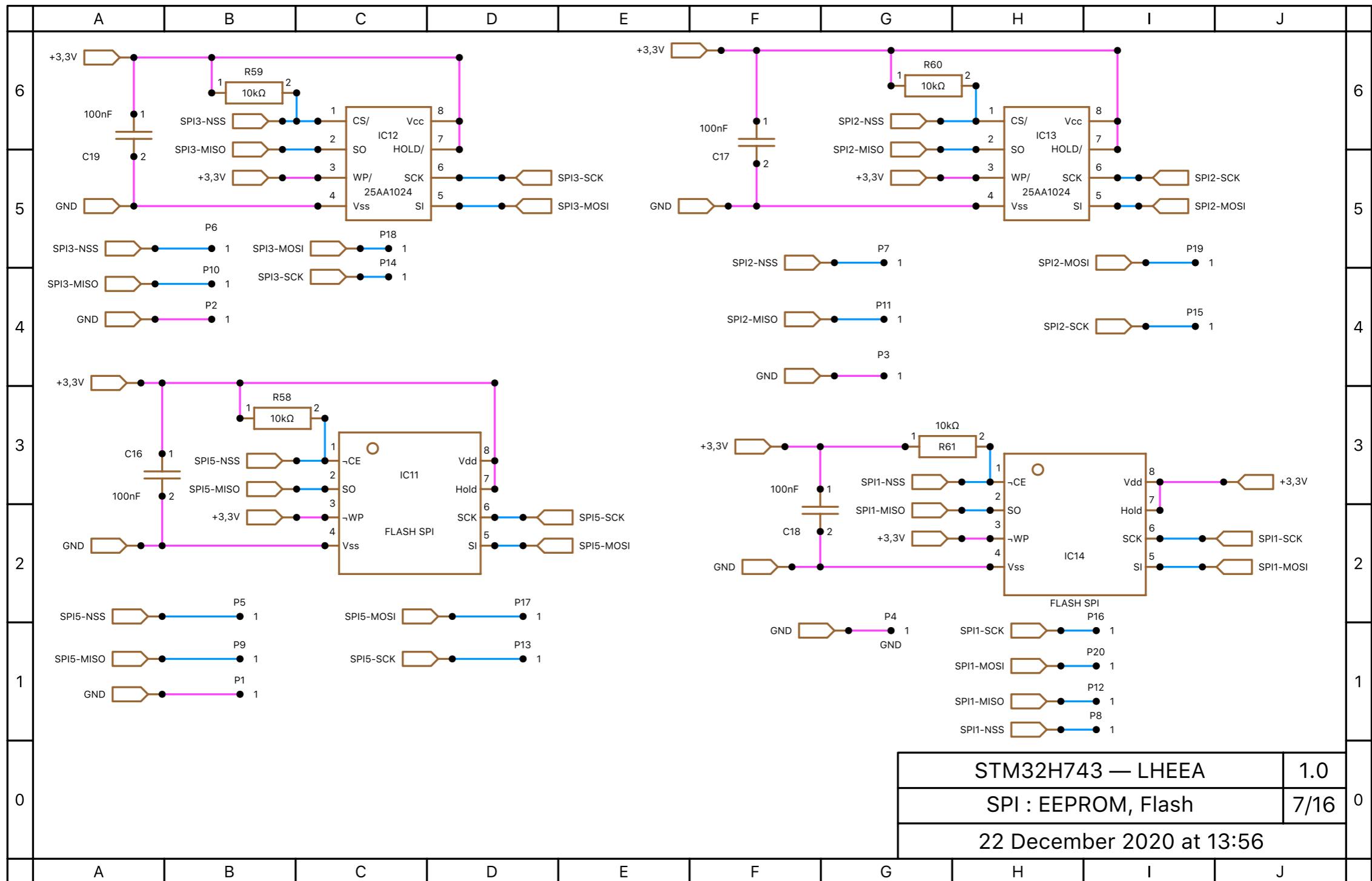
Plan de la carte (5/16)



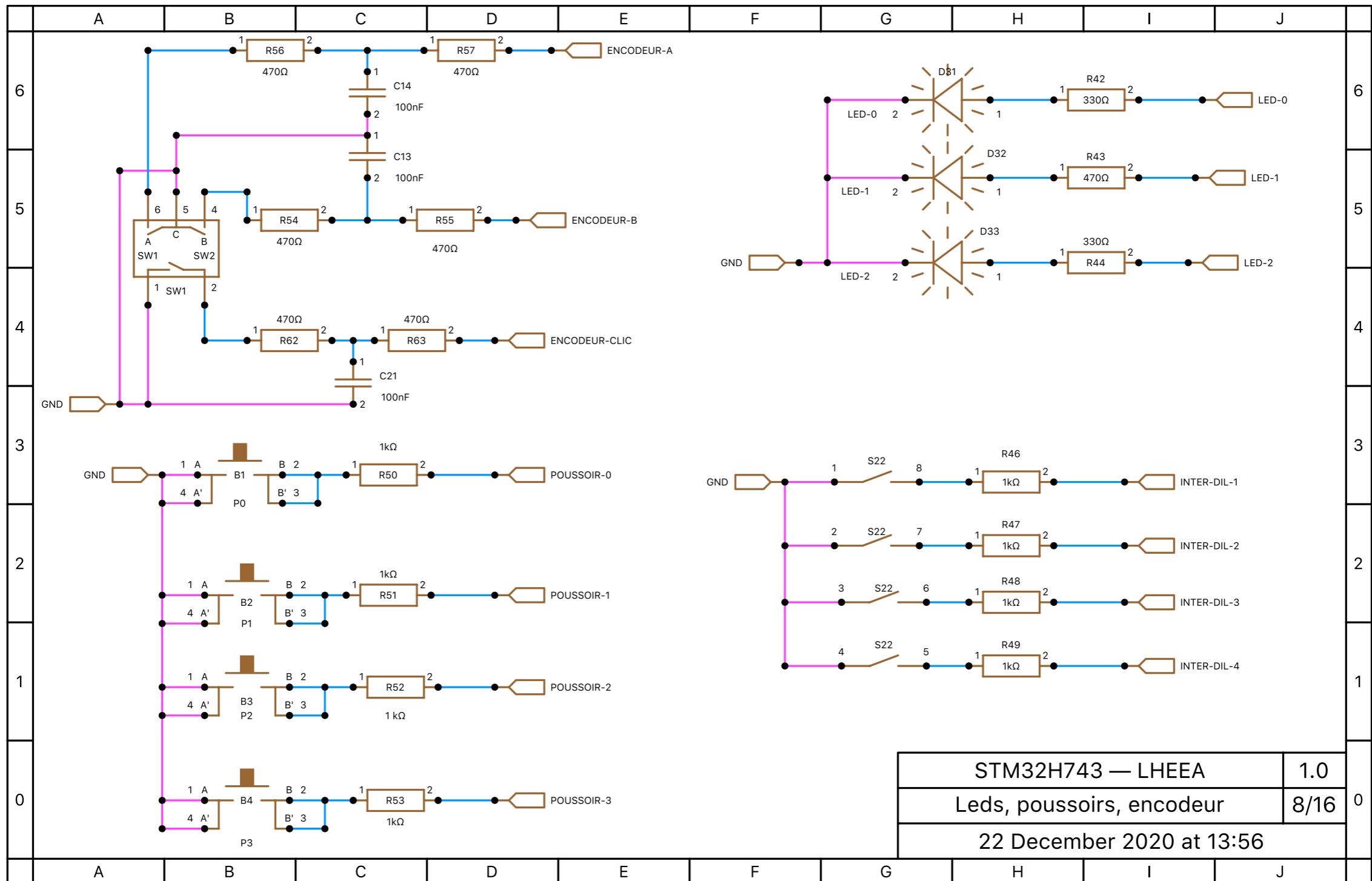
Plan de la carte (6/16)



Plan de la carte (7/16)



Plan de la carte (8/16)

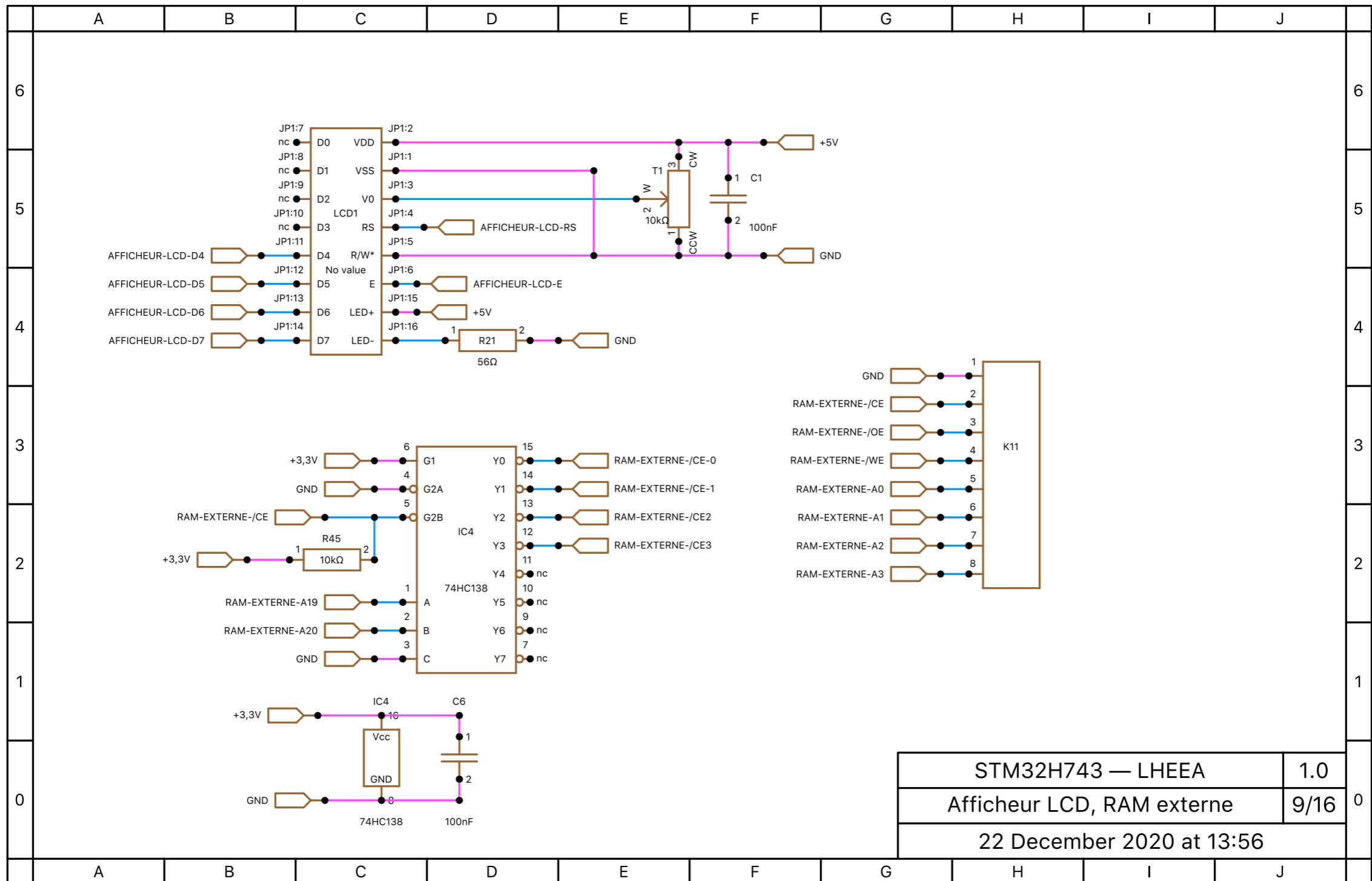


STM32H743 — LHEEA

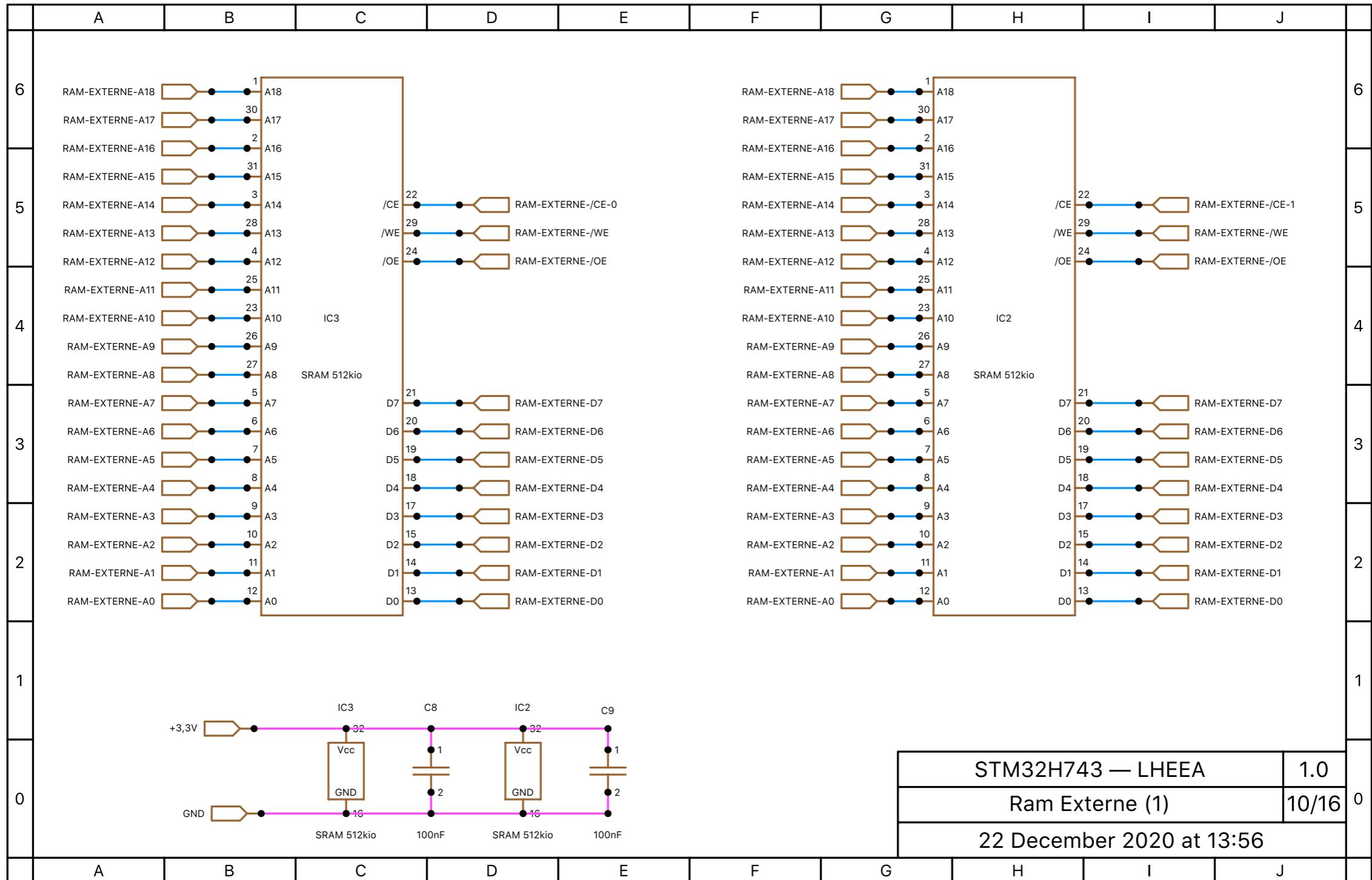
Leds, poussoirs, encodeur

22 December 2020 at 13:56

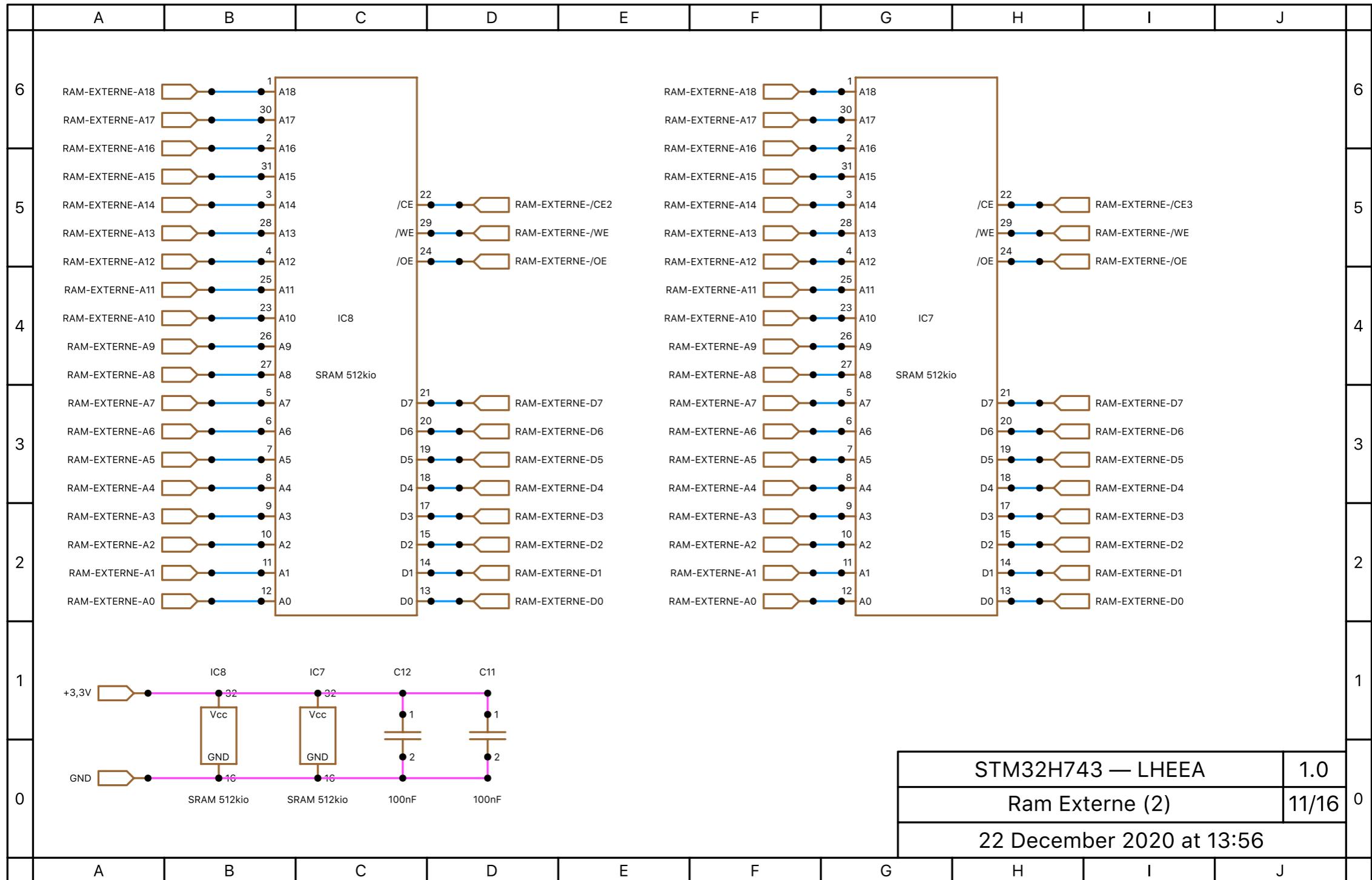
Plan de la carte (9/16)



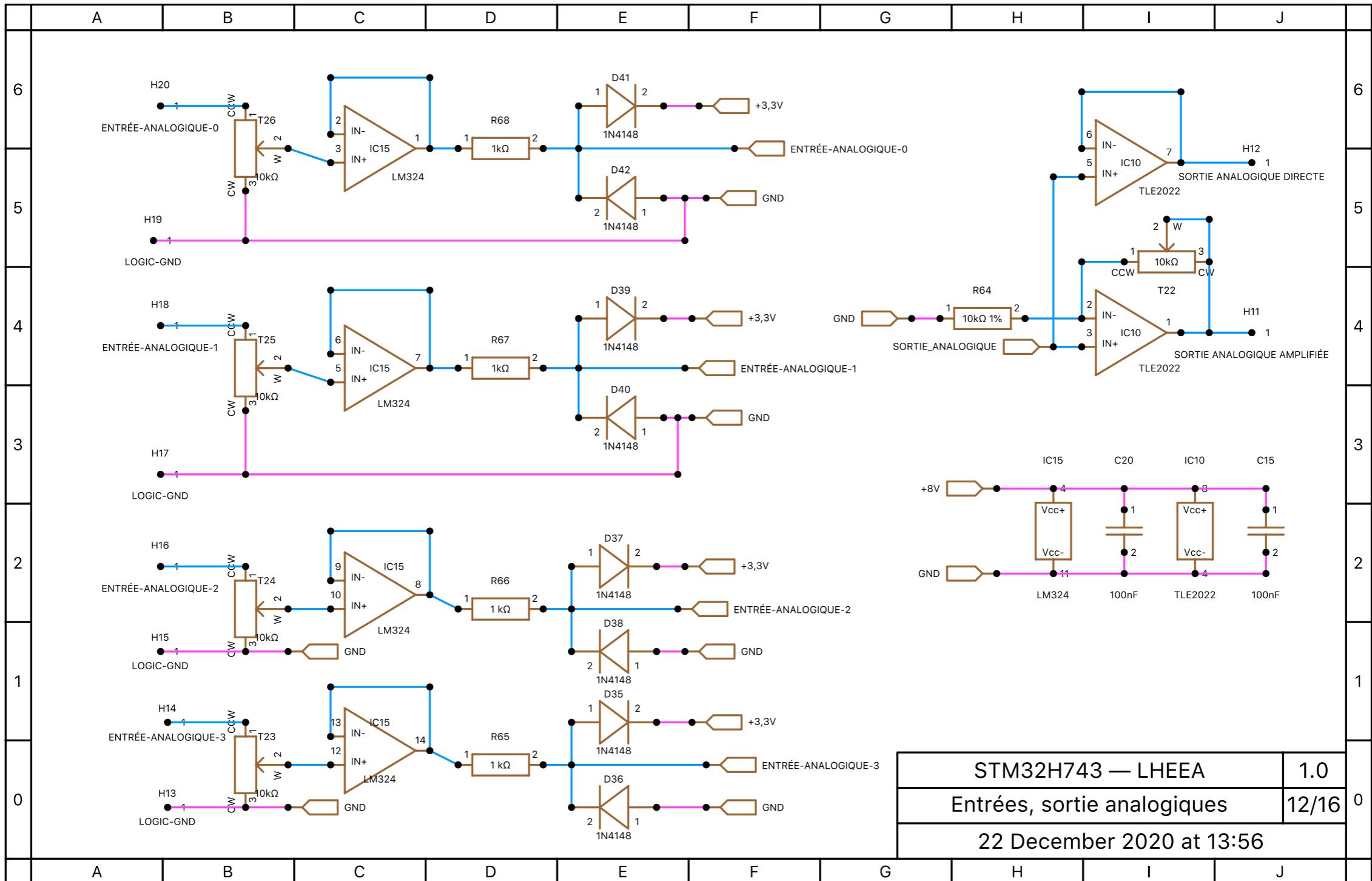
Plan de la carte (10/16)



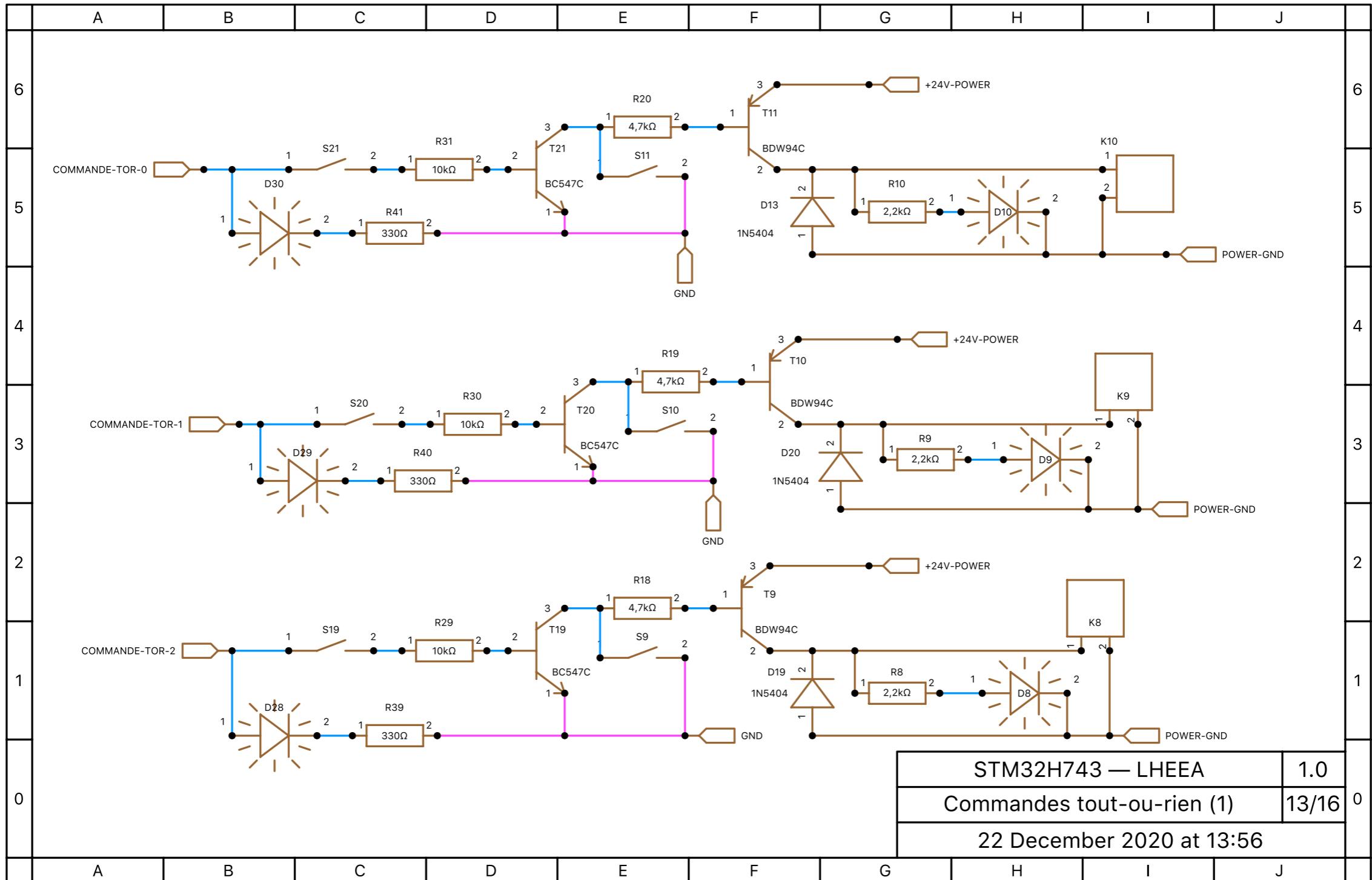
Plan de la carte (11/16)



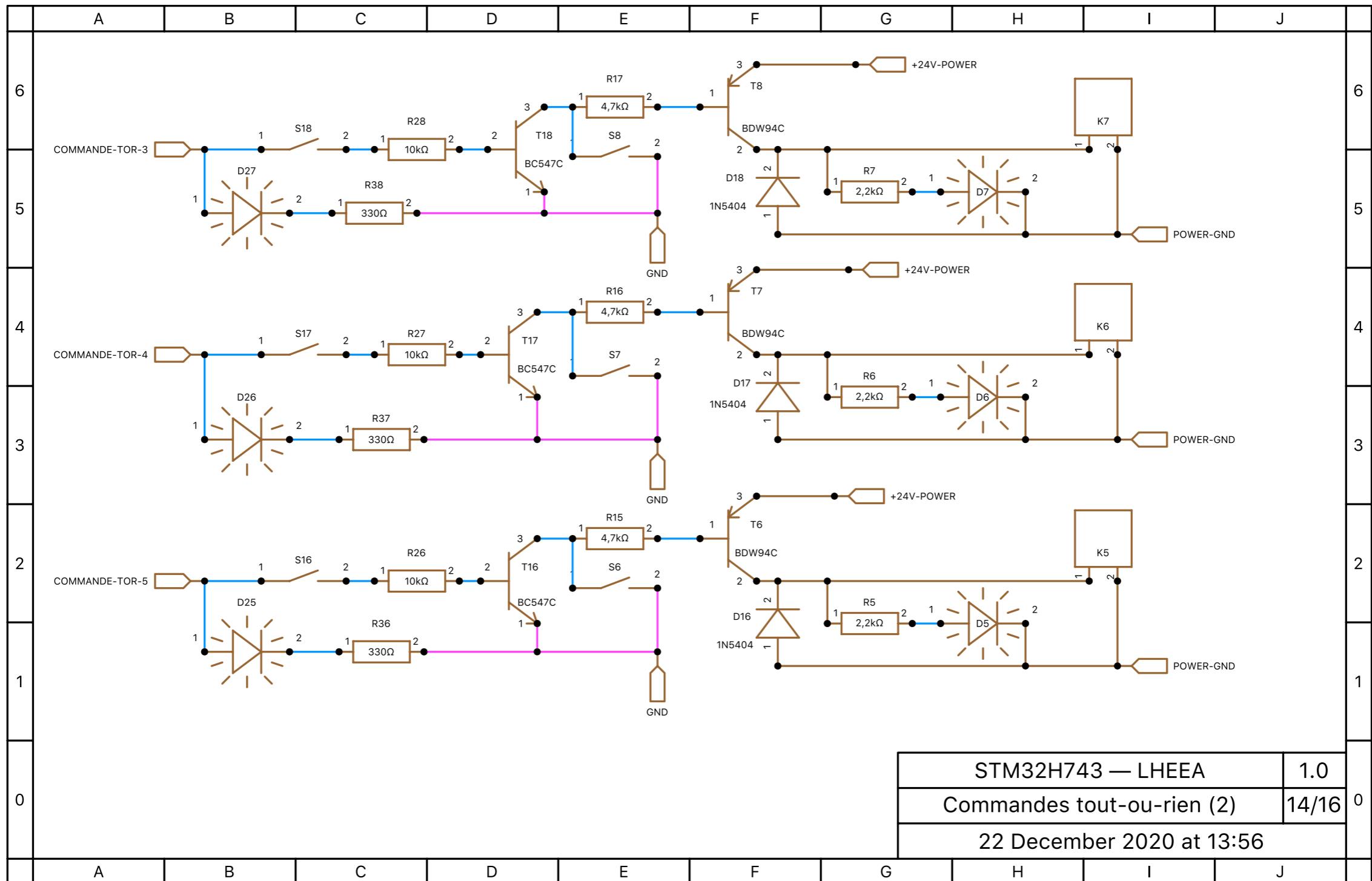
Plan de la carte (12/16)



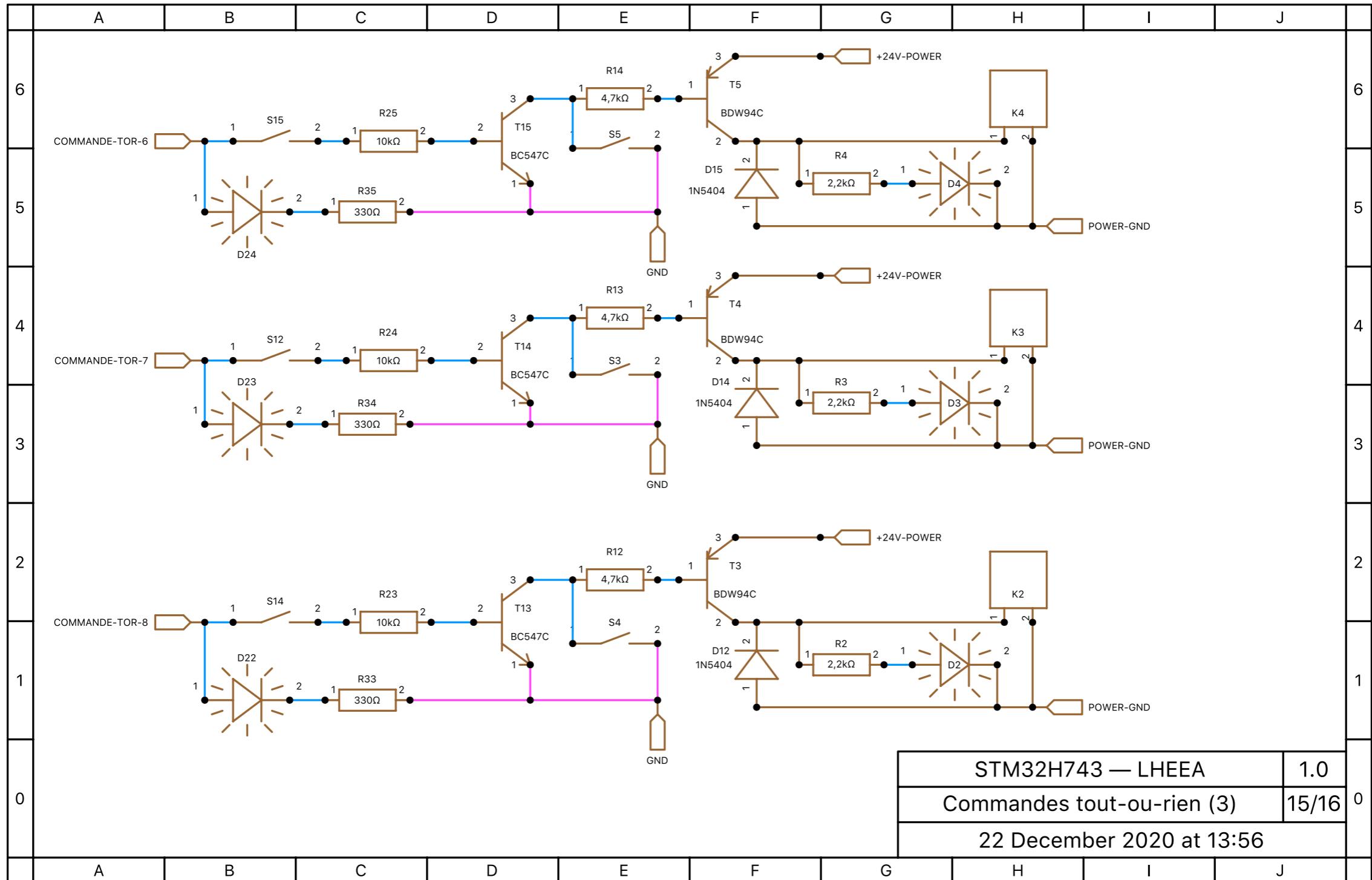
Plan de la carte (13/16)



Plan de la carte (14/16)



Plan de la carte (15/16)



Plan de la carte (16/16)

