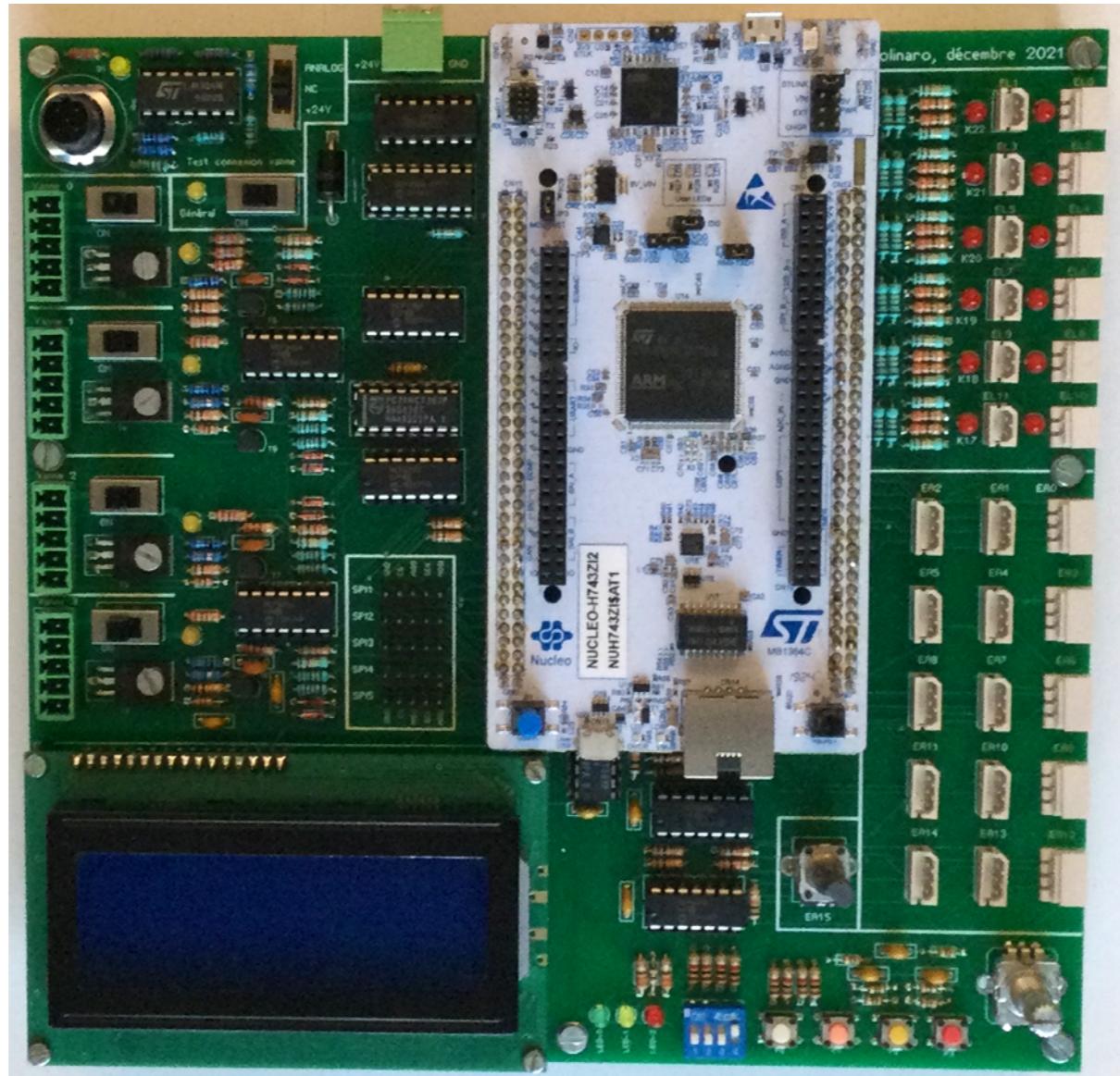


Mode d'emploi de la carte

STM32H743 LS2N



A

Introduction

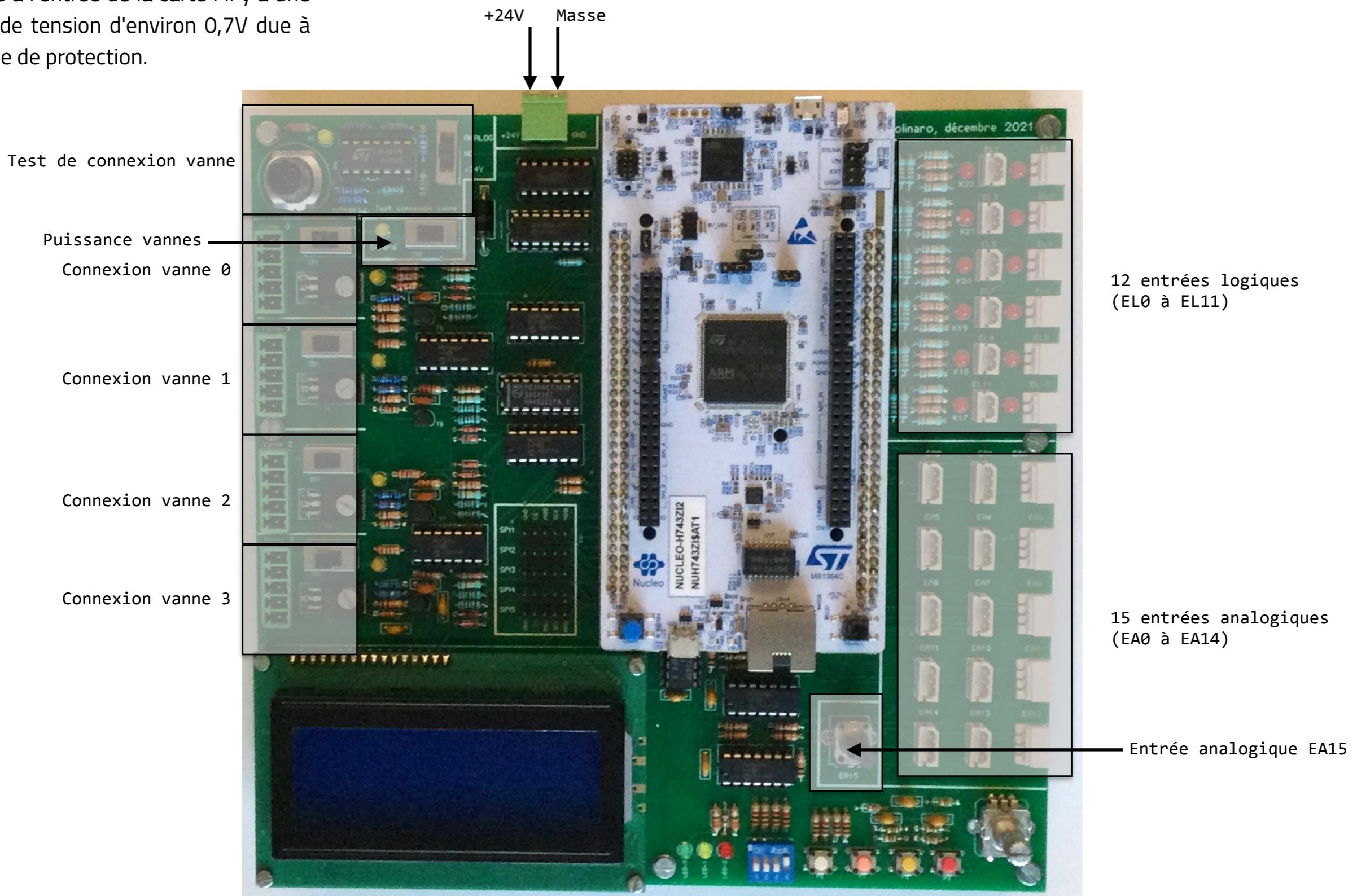
Carte *STM32H743 LS2N*

Le fichier *E/Canari* 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-ls2n>

Schéma de connexion

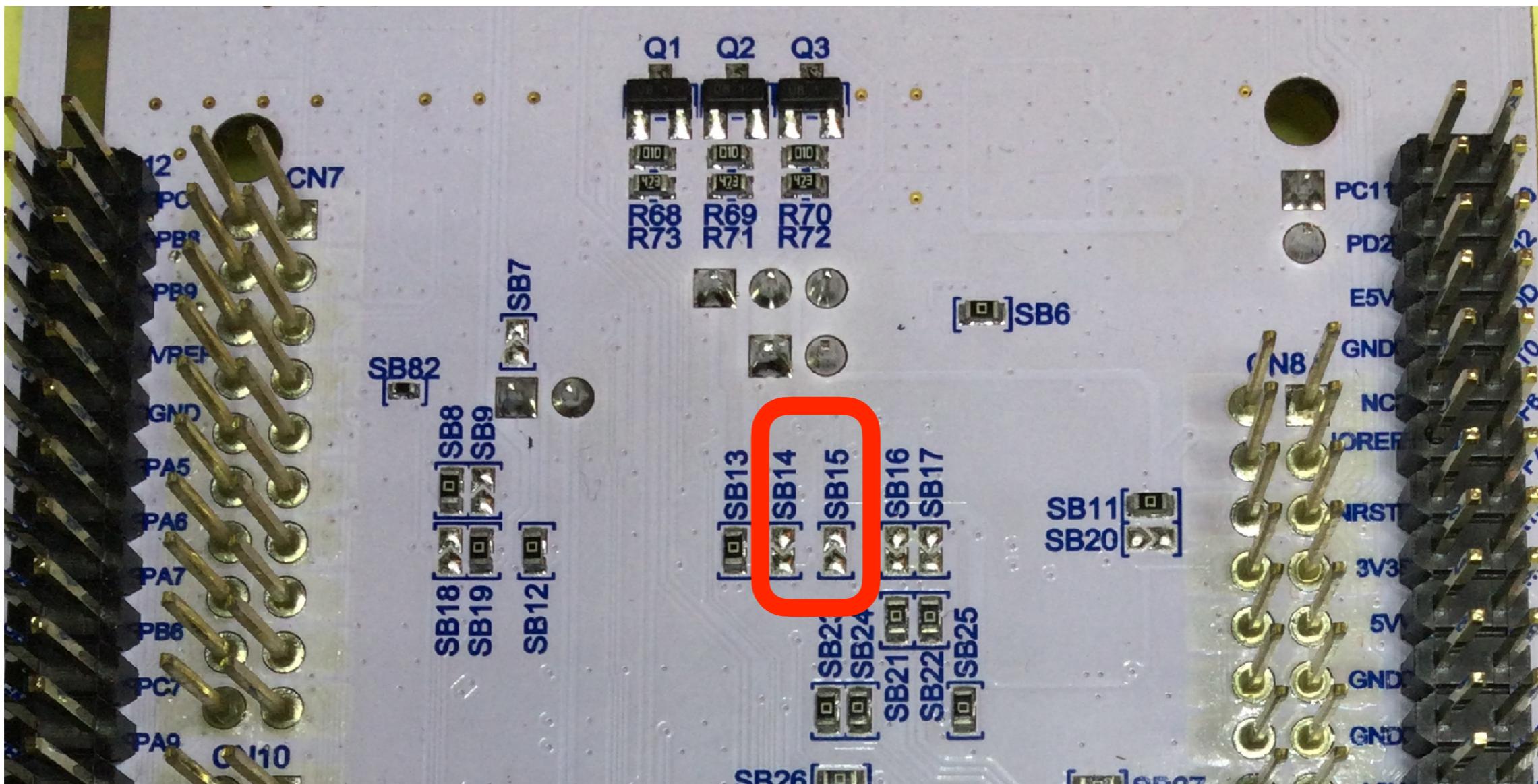
Attention. La tension délivrée à une vanne n'est pas exactement la tension fournie à l'entrée de la carte : il y a une chute de tension d'environ 0,7V due à la diode de protection.

Il y a une diode de protection, qui protège carte et vannes si l'alimentation est inversée.



Avertissement

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**.



B

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 `configurerCarteH743LS2N` (définie dans `STM32H743-configuration-1s2n.cpp`), à appeler dans `setup`.

La variable à utiliser est `lcd` (déclarée dans `STM32H743-configuration-1s2n.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

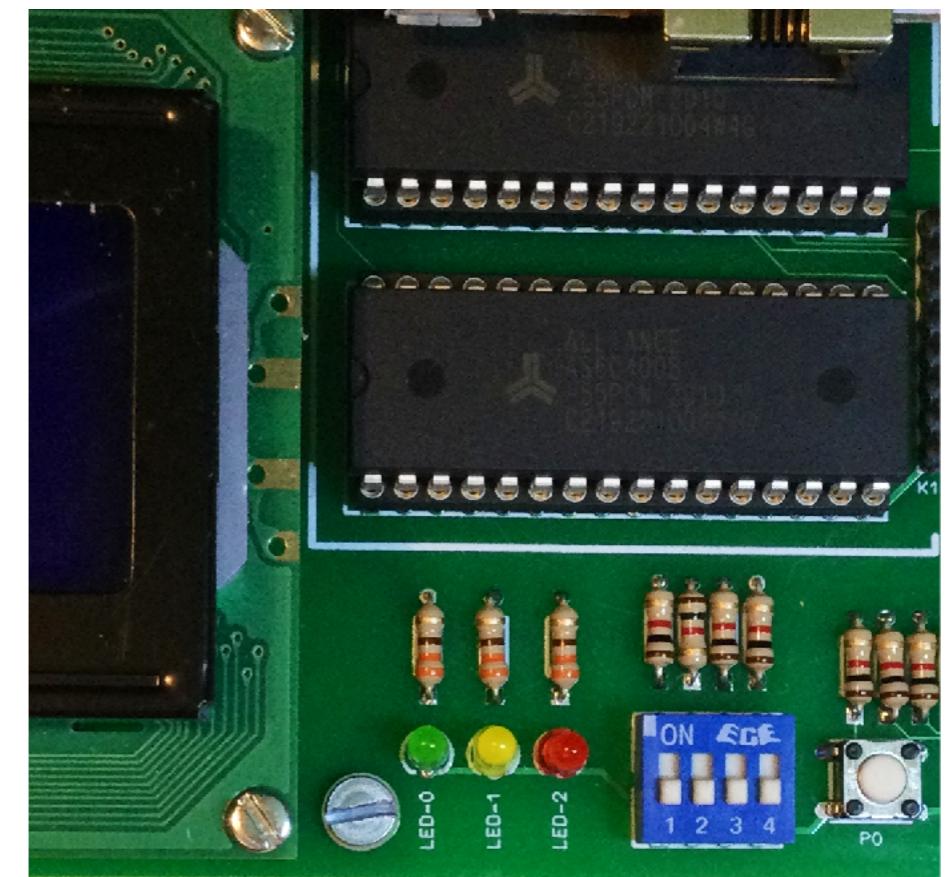
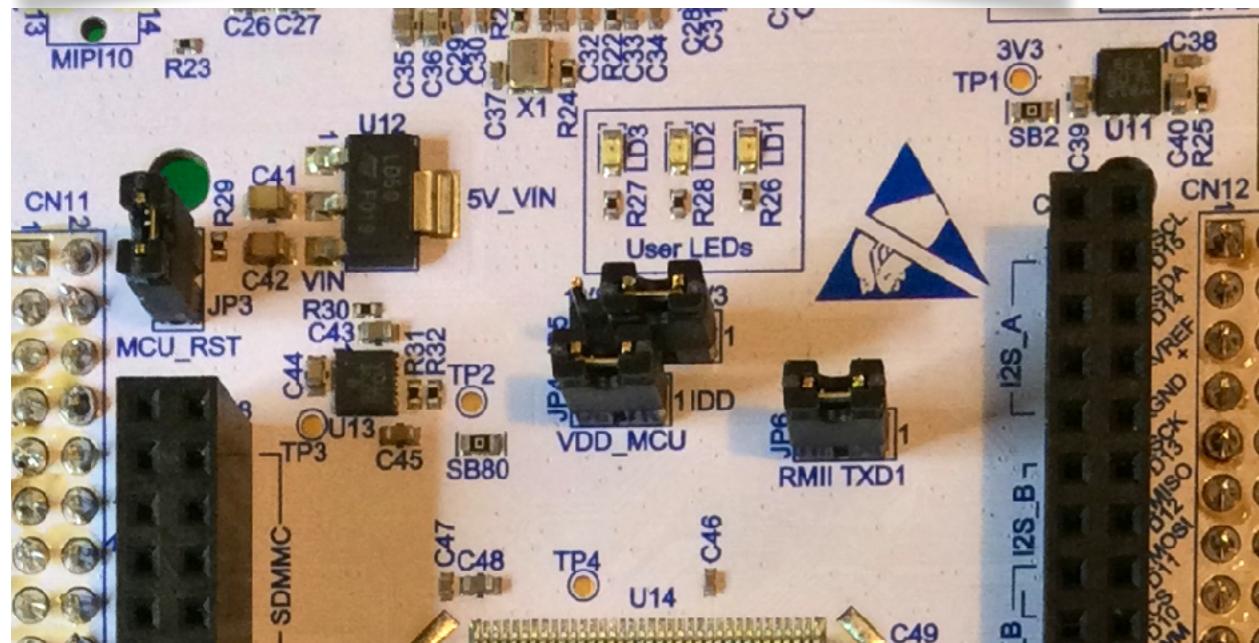
Il y a 6 leds disponibles :

- 3 leds sur la carte Nucleo ;
 - 3 leds sur la carte LS2N.

À chacune correspond un nom (déclaré dans STM32H743-configuration-1s2n.h), ce qui permet de les identifier plus facilement. La fonction configurerCarteH743LS2N (définie dans STM32H743-configuration-1s2n.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_VERT = PA0 ;
static const uint8_t LED_1_JAUNE = PA3 ;
static const uint8_t LED_2_ROUGE = PB2 ;
static const uint8_t NUCLEO_LD1_VERT = PB0 ;
static const uint8_t NUCLEO_LD2_JAUNE = PE1 ;
static const uint8_t NUCLEO_LD3_ROUGE = PB14 ;
```

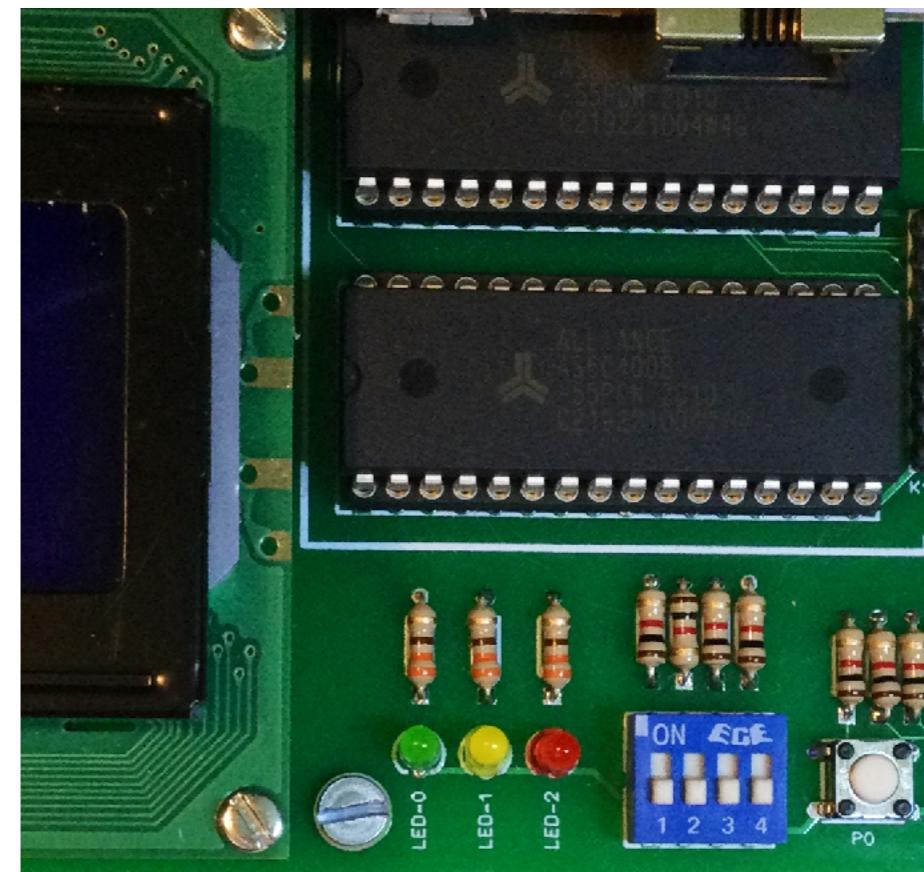


Interrupteurs DIL

Il y a 4 interrupteurs DIL sur la carte.

À chacun correspond un nom (déclaré dans STM32H743-configuration-1s2n.h), ce qui permet de les identifier plus facilement. La fonction configurerCarteH743LS2N (définie dans STM32H743-configuration-1s2n.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.

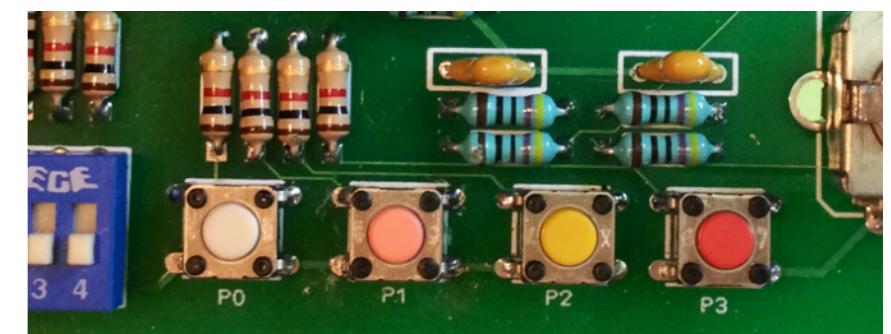
Poussoirs

Il y a 5 poussoirs :

- un poussoir (bleu) sur la carte Nucleo ;
- quatre poussoirs (blanc, rose, jaune, rouge) sur la carte LS2N.

À chacun correspond un nom (déclaré dans STM32H743-configuration-1s2n.h), ce qui permet de les identifier plus facilement. La fonction configurerCarteH743LS2N (définie dans STM32H743-configuration-1s2n.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 :

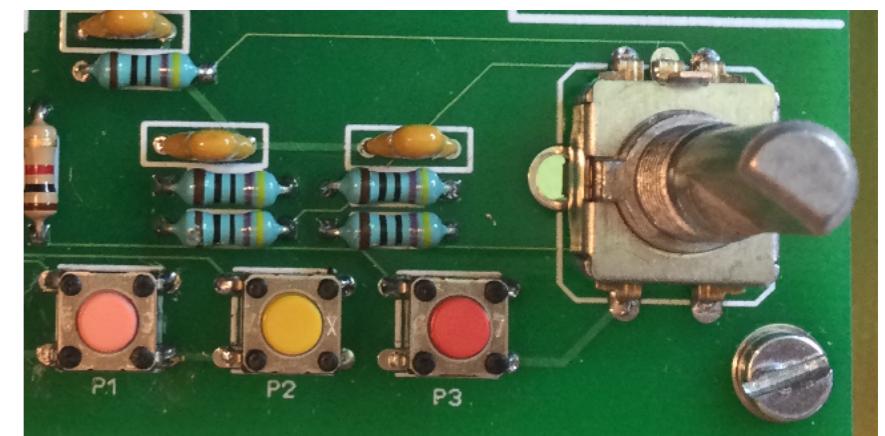
- la première, similaire à celle d'un poussoir ;
- la seconde est la rotation.

Poussoir. Pour la fonction poussoir, utiliser la constante ENCODEUR_CLIC déclarée dans STM32H743-configuration-1s2n.h. La fonction configurerCarteH743LS2N (définie dans STM32H743-configuration-1s2n.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]` :

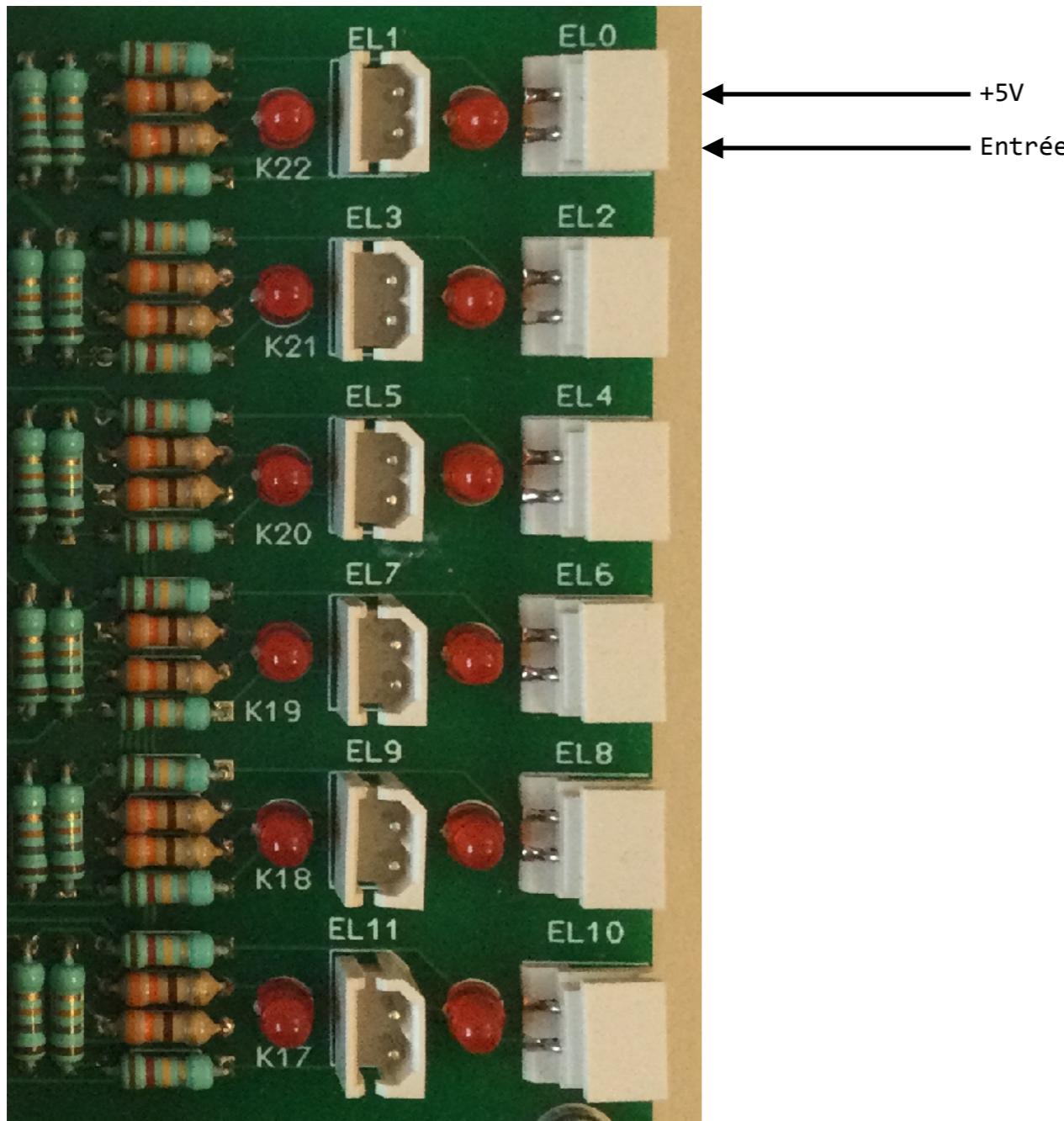
- une rotation dans le sens trigonométrique décrémente la valeur renvoyée, jusqu'à saturer à `borneInf` ;
- une rotation dans le sens des aiguilles d'une montre incrémente la valeur renvoyée, jusqu'à saturer à `borneSup`.

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

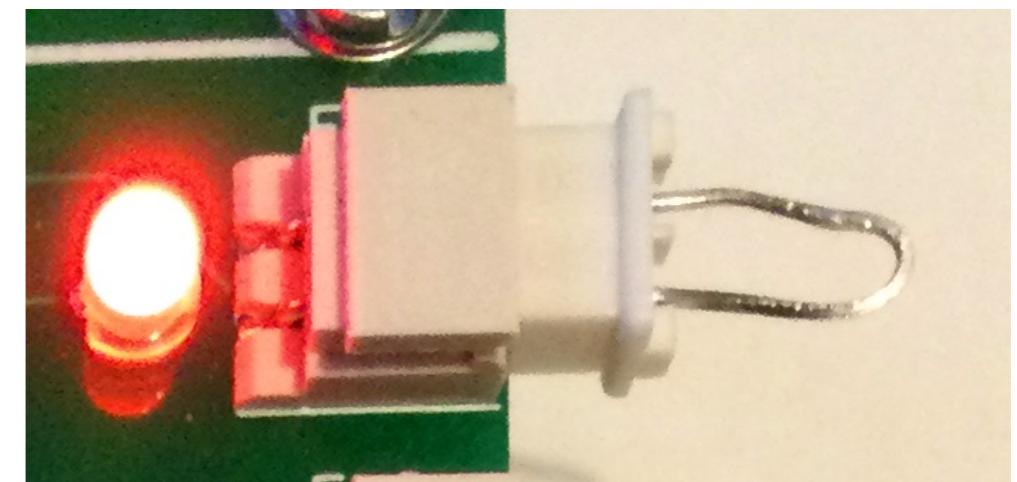
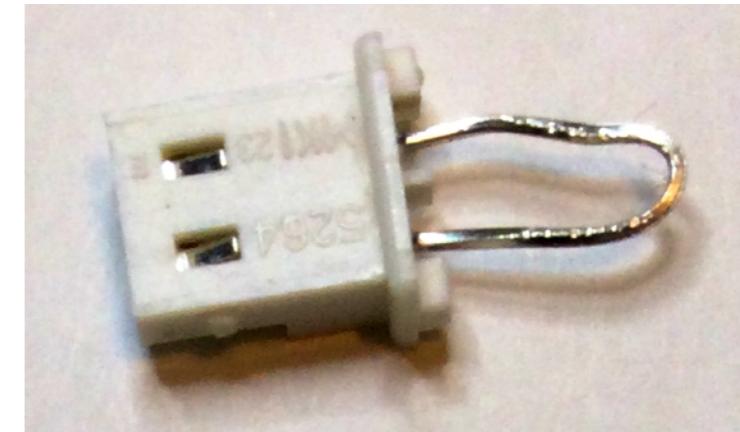
C

Entrées logiques

Les 12 entrées logiques EL0 à EL11



Pour tester une entrée, on peut utiliser un bouchon qui relie l'entrée au +5V.



Connectique

La connectique des entrées logiques est du type SPOX 2,5 mm.



Socle droit

<https://www.tme.eu/fr/details/mx-5267-02a/connecteurs-de-signal-pas-2-50mm/molex/22-03-5025/>



Socle coudé

<https://www.tme.eu/fr/details/mx-5268-02a/connecteurs-de-signal-pas-2-50mm/molex/22-05-7025/>



Prise femelle

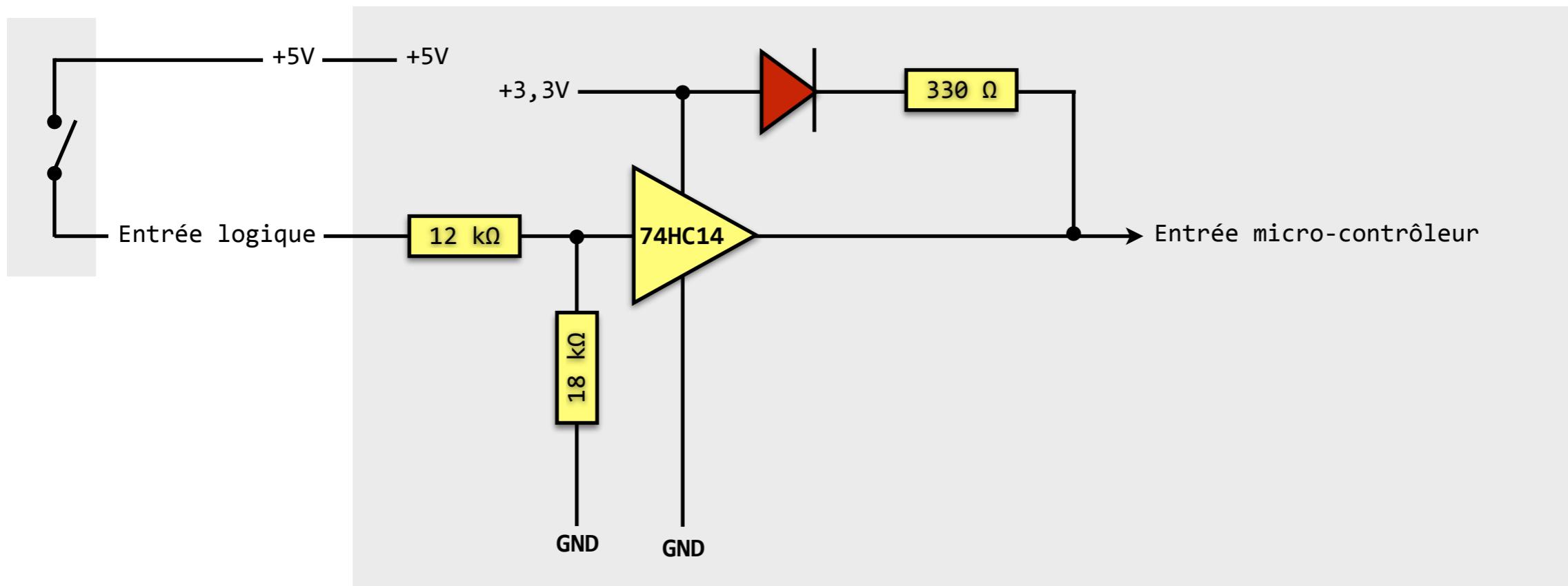
<https://www.tme.eu/fr/details/mx-5264-02/connecteurs-de-signal-pas-2-50mm/molex/50-37-5023/>



Contact (aussi utilisé pour les connecteurs des entrées analogiques)

<https://www.tme.eu/fr/details/mx-5263-pbtl/connecteurs-de-signal-pas-2-50mm/molex/08-70-1040/>

Schéma d'une entrée logique



Par défaut, quand l'entrée est non connectée, l'entrée du micro-contrôleur est à 1, le led est éteinte. Un interrupteur extérieur doit être placé entre le +5V et l'entrée logique. Lorsqu'il est fermé, l'entrée du micro-contrôleur est à 0, le led est allumée.

Lecture d'une entrée logique

L'initialisation des ports correspondants aux entrées logiques est effectuée par la fonction **configurerCarteH743LS2N** (définie dans STM32H743-configuration-1s2n.cpp), à appeler dans setup.

Pour lire une entrée logique individuelle, utiliser **digitalRead**. L'affectation des ports est la suivante :

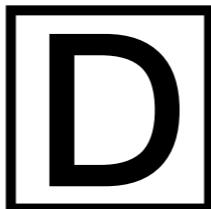
```
static const uint8_t PORT_EL0 = PF0 ;  
static const uint8_t PORT_EL1 = PF1 ;  
static const uint8_t PORT_EL2 = PF2 ;  
static const uint8_t PORT_EL3 = PF3 ;  
static const uint8_t PORT_EL4 = PF4 ;  
static const uint8_t PORT_EL5 = PF5 ;  
static const uint8_t PORT_EL6 = PF10 ;  
static const uint8_t PORT_EL7 = PF11 ;  
static const uint8_t PORT_EL8 = PF12 ;  
static const uint8_t PORT_EL9 = PF13 ;  
static const uint8_t PORT_EL10 = PF14 ;  
static const uint8_t PORT_EL11 = PF15 ;
```

Attention, la valeur renvoyée est **HIGH** (c'est-à-dire 1) si l'entrée est non connectée, et **LOW** (c'est-à-dire 0) si elle est activée.

Pour lire les 12 entrées en une seule fois, appeler **lireToutesEntreesLogiques**. Cette fonction renvoie un entier non signé de 16 bits (**uint16_t**) :

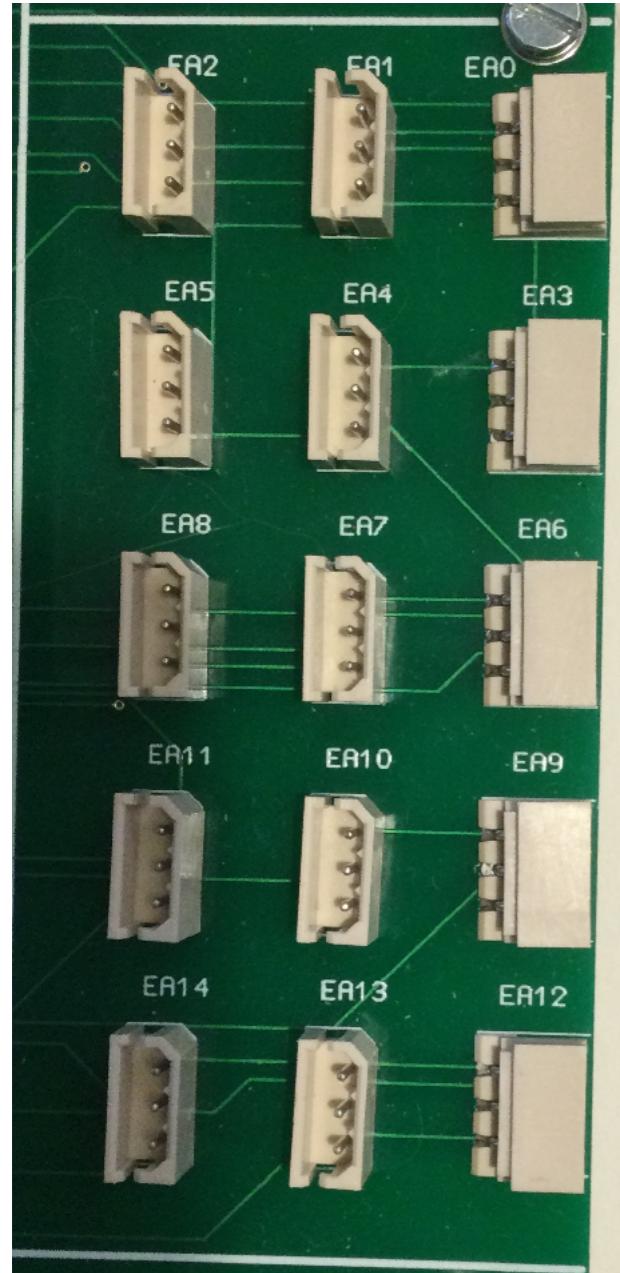
- le bit **i** ($0 \leq i \leq 11$) est à 1 si l'entrée est non connectée, 0 si elle est activée ;
- les bits 12 à 15 sont à 0.

Voir le croquis 04-croquis-test-entrees-logiques pour un exemple d'appel de cette fonction.

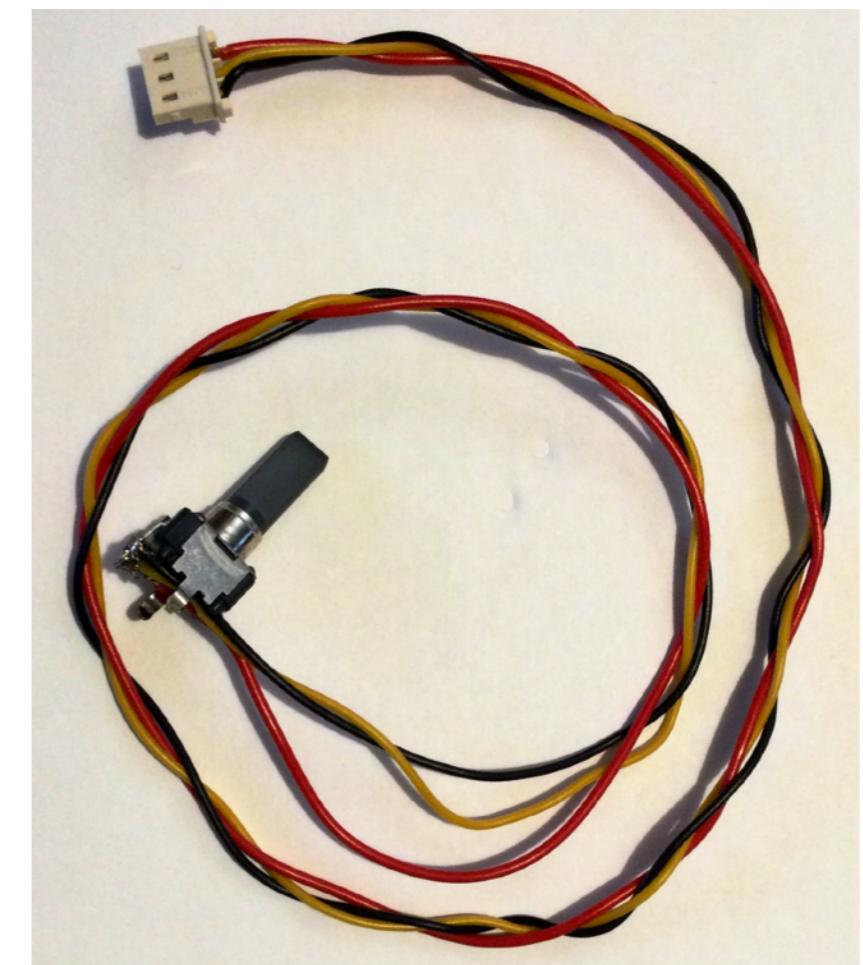
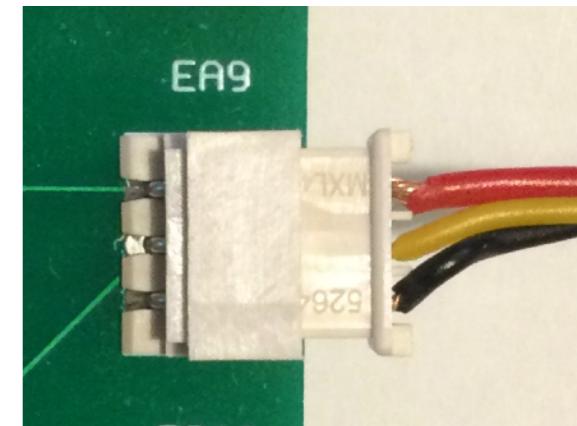


Entrées analogiques

Entrées analogiques EA0 à EA14



+5V
Entrée analogique
0V



Entrée analogique EA15



Ce potentiomètre est relié à l'entrée EA15.

Connectique

La connectique des entrées logiques est du type SPOX 2,5 mm.



Socle droit

<https://www.tme.eu/fr/details/mx-5267-03a/connecteurs-de-signal-pas-2-50mm/molex/22-03-5035/>



Socle coudé

<https://www.tme.eu/fr/details/mx-5268-03a/connecteurs-de-signal-pas-2-50mm/molex/22-05-7035/>



Prise femelle

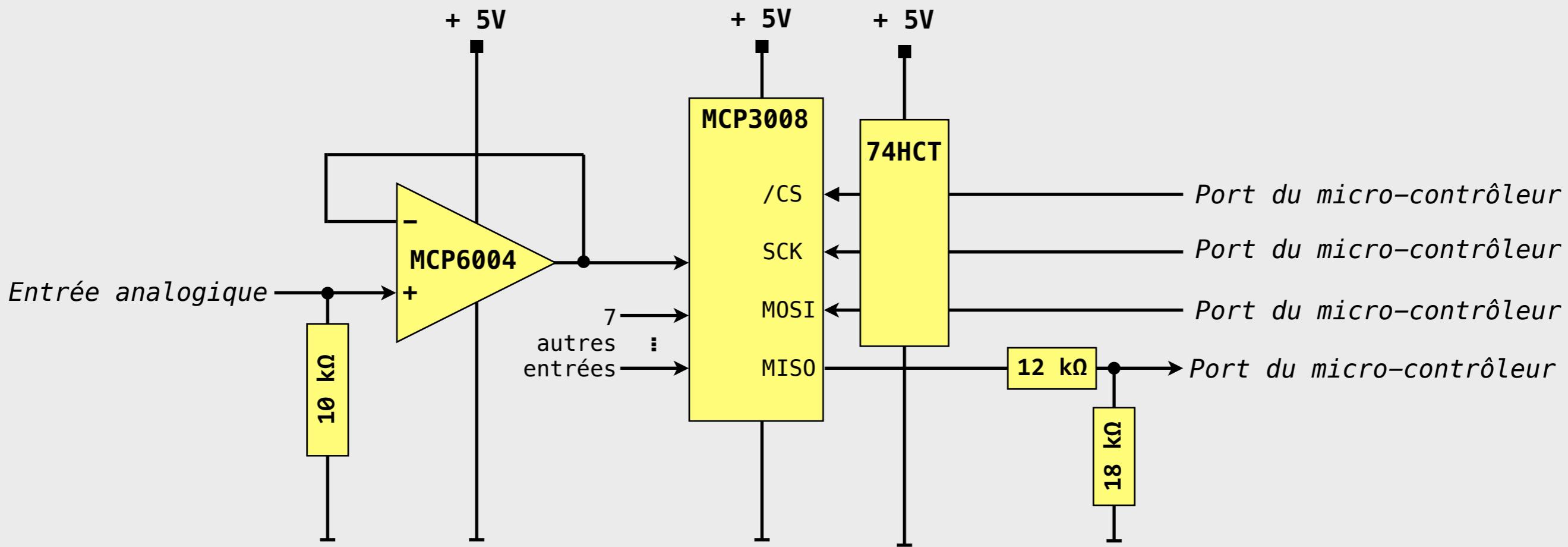
<https://www.tme.eu/fr/details/mx-5264-03/connecteurs-de-signal-pas-2-50mm/molex/50-37-5033/>



Contact (aussi utilisé pour les connecteurs des entrées logiques)

<https://www.tme.eu/fr/details/mx-5263-pbtl/connecteurs-de-signal-pas-2-50mm/molex/08-70-1040/>

Schéma d'une entrée analogique 5V



Le MCP6004 est un quadruple amplificateur opérationnel *rail-to-rail*, alimenté sous 5V. Il sert d'adaptation d'impédance pour le MCP3008 et de protection. La résistance de 10 kΩ amène l'entrée à zéro en l'absence de connexion.

Le MCP3008 est un octuple convertisseur analogique -> numérique 10 bits. Il est alimenté sous 5V (sa rapidité augmente avec la tension). L'adaptation de tension avec le micro-contrôleur alimenté sous 5V se fait avec un pont diviseur pour MISO (entrée micro-contrôleur), et par un circuit de la famille 74HCT (74HCT08, 74HCT32) pour les sorties du micro-contrôleur.

Remarque sur le sens de rotation des potentiomètres

Normalement, tourner un potentiomètre dans le sens des aiguilles d'une montre provoque une augmentation de l'entrée.

Le potentiomètre **EA15** est mal connecté, le tourner dans le sens des aiguilles d'une montre provoque une diminution de l'entrée.

Le cordon de test est câblé comme le potentiomètre **EA15**.

Lire une entrée analogique (1/2)

Il y a deux fonctions possibles **lireEntreeAnalogique_9bits** et **lireEntreeAnalogique_10bits** (voir page suivante).

```
uint16_t lireEntreeAnalogique_9bits (const uint8_t inIndiceEntree) ;
```

La fonction **lireEntreeAnalogique_9bits** a un argument : **inValue**, un entier non signé sur 8 bits qui code la valeur analogique :

- **inIndiceEntree** entre 0 et 15 -> la valeur renvoyée est l'acquisition de l'entrée **inIndiceEntree** ;
- **inIndiceEntree** > 15 -> la valeur renvoyée est 0.

La valeur renvoyée est proportionnelle à la tension de l'entrée :

- tension 0V -> valeur renvoyée est 0 ;
- tension 5V -> la valeur renvoyée est 511.

Par exemple, si la tension de l'entrée est 2,33V : la valeur renvoyée est $2,33V * 511 / 5V = 238$.

Pour obtenir la tension en Volt à partir de la valeur lue, il faut multiplier par $5V / 511$:

```
tension_en_volt = float (lireEntreeAnalogique_9bits (inIndiceEntree)) * 5.0 / 511.0 ;
```

Voir le croquis d'exemple 05-croquis-test-entrees-analogiques.

Lire une entrée analogique (2/2)

Il y a deux fonctions possibles `lireEntreeAnalogique_9bits` (voir page précédente) et `lireEntreeAnalogique_10bits`.

```
uint16_t lireEntreeAnalogique_10bits (const uint8_t inIndiceEntree) ;
```

La fonction `lireEntreeAnalogique_10bits` a un argument : `inValue`, un entier non signé sur 8 bits qui code la valeur analogique :

- `inIndiceEntree` entre 0 et 15 -> la valeur renvoyée est l'acquisition de l'entrée `inIndiceEntree` ;
- `inIndiceEntree > 15` -> la valeur renvoyée est 0.

La valeur renvoyée est proportionnelle à la tension de l'entrée :

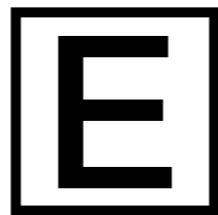
- tension 0V -> valeur renvoyée est 0 ;
- tension 5V -> la valeur renvoyée est 1023.

Par exemple, si la tension de l'entrée est 2,33V : la valeur renvoyée est $2,33V * 1023 / 5V = 477$.

Pour obtenir la tension en Volt à partir de la valeur lue, il faut multiplier par $5V / 1023$:

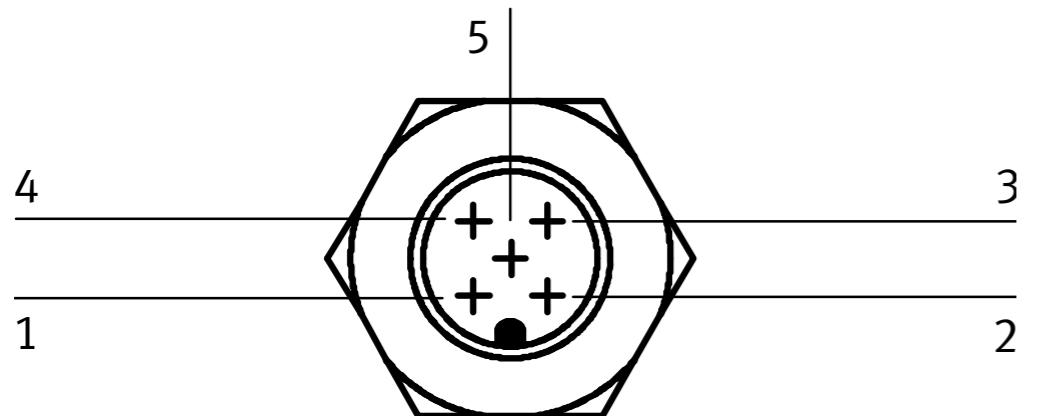
```
tension_en_volt = float (lireEntreeAnalogique_10bits (inIndiceEntree)) * 5.0 / 1023.0 ;
```

Voir le croquis d'exemple 05-croquis-test-entrees-analogiques.



Contrôle des vannes

Câble de connexion d'une vanne



Broche n°	Couleur ¹⁾	VPPE
1	ROUGE	+24 V CC
2	BLEU	Entrée analogique – valeur de consigne / – w
3	NOIR	GND
4	MARRON	Entrée analogique + valeur de consigne / + w (0 ... 10 V ou 4 ... 20 mA)
5	JAUNE	Sortie de commutation (24 V) ou analogique de commutation (0 ... 10 V ou 4 ... 20 mA) + valeur réelle / X _{OUT}

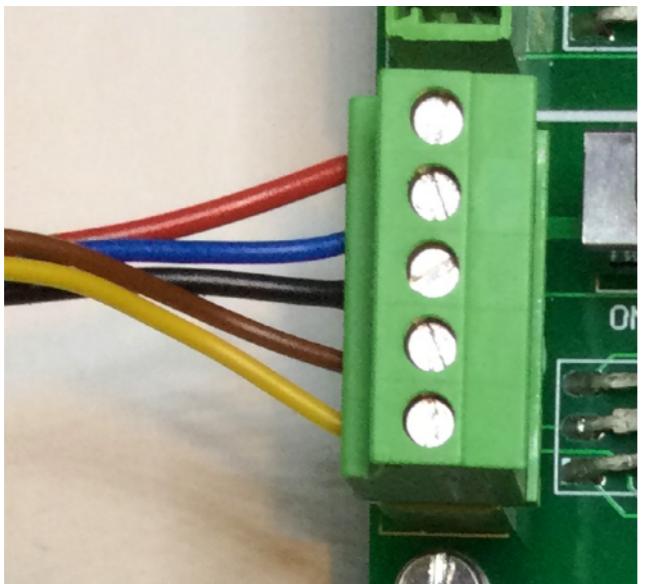
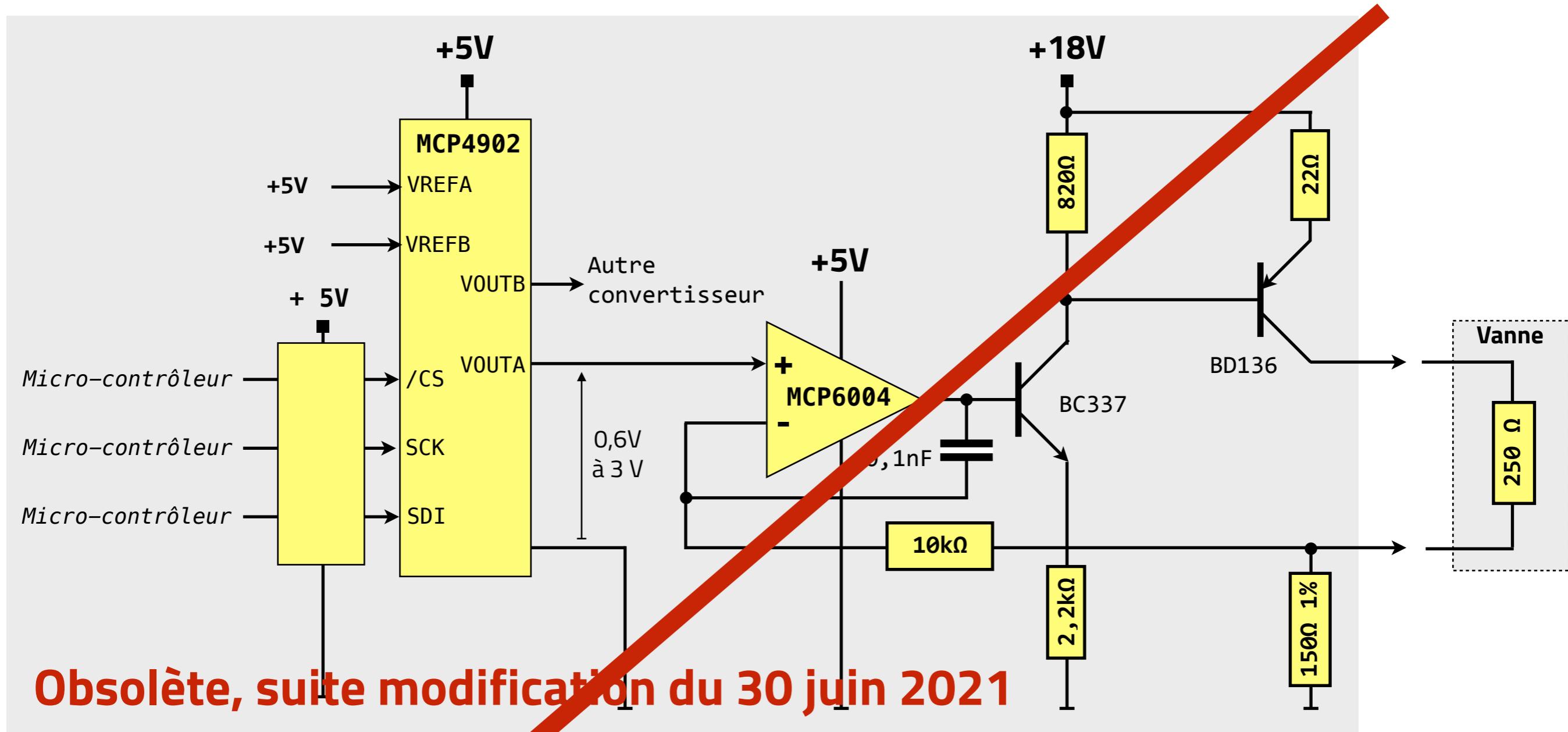


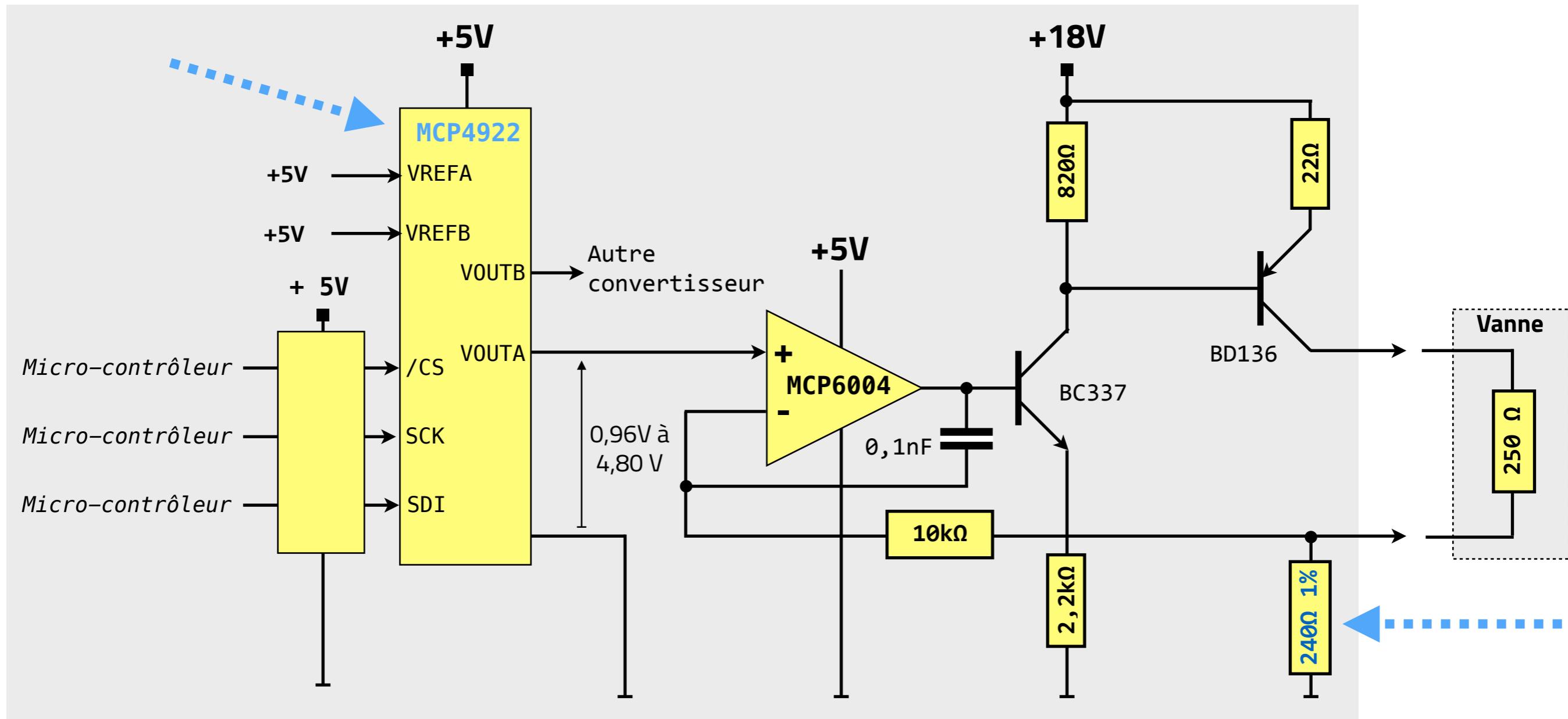
Schéma du convertisseur 4 / 20 mA



Le MCP4902 est un double convertisseur numérique -> analogique 8 bits. Un composant de la série 74HCT assure l'adaptation de tension. Le facteur de conversion est calculé comme suit :

$$courant_mA = inCommande * \frac{5\text{ V}}{255} * \frac{1000\text{ mA/A}}{150\text{ }\Omega} = \frac{inCommande}{7.65}$$

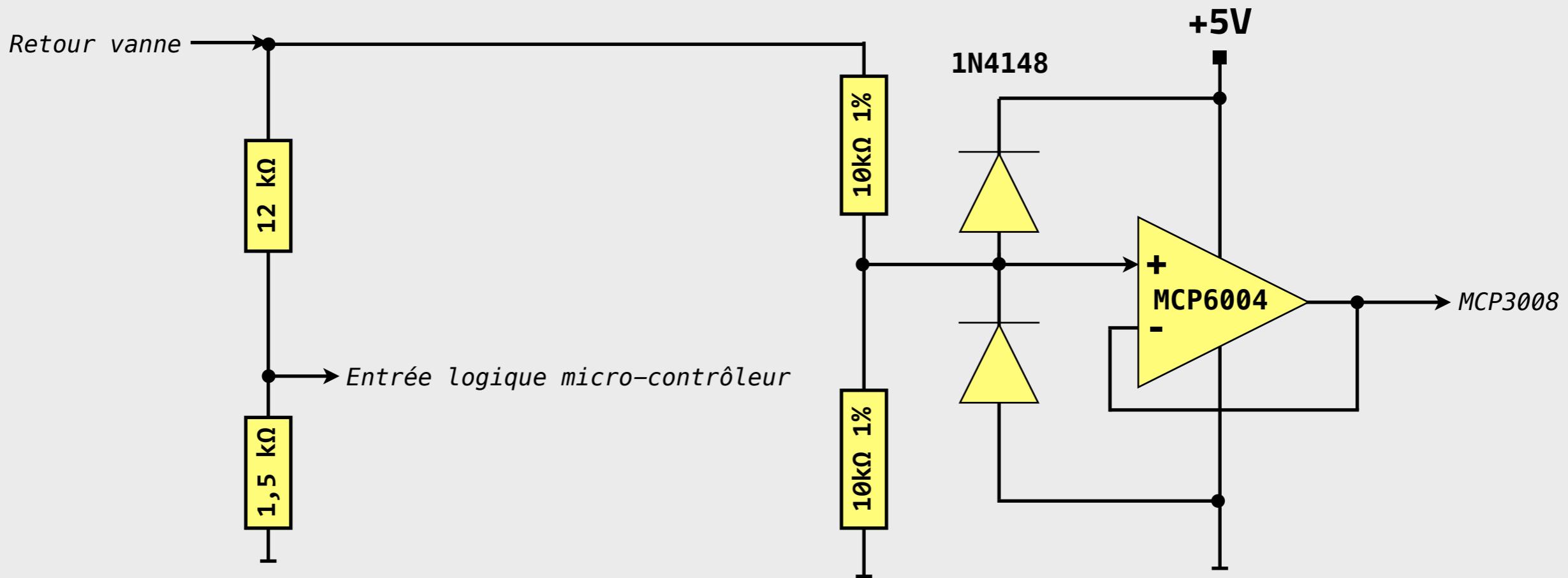
Schéma du convertisseur 4 / 20 mA [30 juin 2022]



Le **MCP4922** est un double convertisseur numérique -> analogique 12 bits. Un composant de la série 74HCT assure l'adaptation de tension. Le facteur de conversion est calculé comme suit :

$$\text{courant_mA} = \text{inCommande} * \frac{5 \text{ V}}{4095} * \frac{1000 \text{ mA/A}}{240 \Omega} = \frac{\text{inCommande}}{196.56}$$

Schéma de la connexion du retour de la vanne



La vanne peut être programmée pour fournir soit une tension logique 0 / 24 V, soit une tension analogique 0 / 10V. Les deux diodes 1N4148 protègent l'amplificateur opérationnel lorsque le niveau logique haut (24V) est délivré.

La valeur retournée par le convertisseur MCP3008 est :

$$retour_analogique = \frac{tension_en_volt}{2} * \frac{511}{5V} = tension_en_volt * 51.1$$

Connecteurs



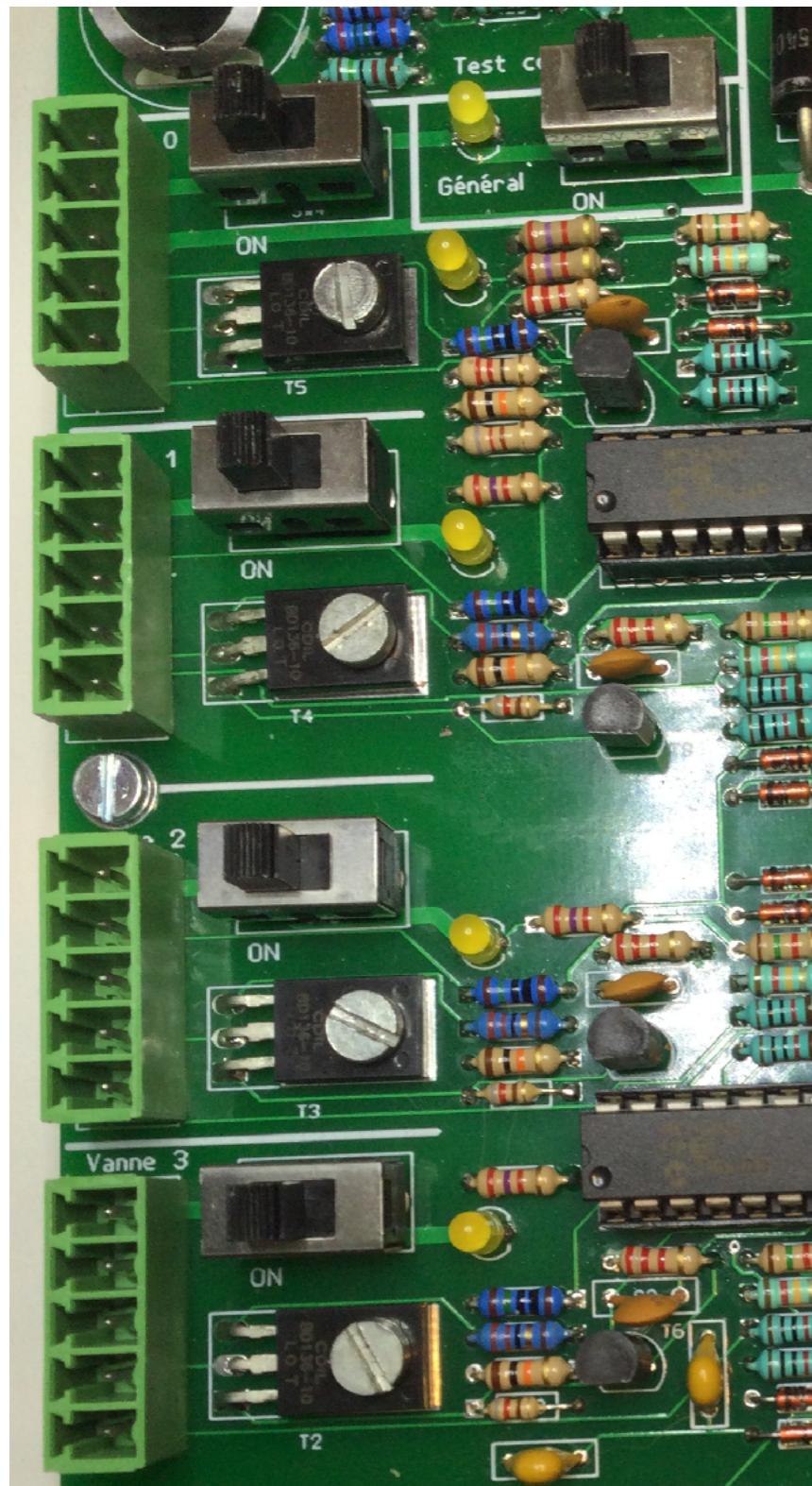
Connecteur **Amphenol SBP-05B5FFA-SL8001**, référence TME :

<https://www.tme.eu/fr/details/sbp-05bffa-sl8001/fiches-et-prises-m12/amphenol/msbp-05bffa-sl8001/>

Bornier enfichable **Phoenix Contact**, *3.81 mm, 5 Voies, 28AWG à 16AWG, 1.5 mm², Par vis, 8 A*, référence Farnell :

<https://fr.farnell.com/phoenix-contact/1827156/fiche-femelle-90deg-ci-5v-3-81mm/dp/3705020>

Commande de puissance



5 interrupteurs à glissière permettent de contrôler la tension (fil rouge) envoyée à chaque vanne :

- un interrupteur général, commun au quatre vannes ;
- pour chaque vanne, un interrupteur particulier.

Une vanne est alimentée en +24V si son interrupteur particulier et l'interrupteur général sont actifs.

Pour chaque interrupteur, la présence de la tension est signalée par une led jaune.

Attention. La tension délivrée à la vanne n'est pas exactement la tension fournie à l'entrée de la carte : il y a une chute de tension d'environ 0,7V due à la diode de protection.

Commande 4 / 20 mA

```
void commandeVanne (const uint32_t inNumeroVanne, // 0 à 3  
                     const uint8_t inCommande) ;
```

La fonction `commandeVanne` présente deux arguments :

- le numéro (0 à 3) de la vanne que l'on veut commander (si l'argument est > 3, l'appel est ignoré)
- `inCommande`, la valeur image du courant de sortie.

La valeur de `inCommande` est calculée comme suit :

```
inCommande = uint8_t (courant_en_mA * 7.65)
```

Par exemple :

- pour 4 mA, la commande est 31 ;
- pour 20 mA, la commande est 153.

Attention, l'électronique de commande est conçue pour assurer un courant proportionnel à la commande dans la plage [4 mA, 20 mA]. En dehors, des non linéarités peuvent apparaître.

Obsolète, suite modification du 30 juin 2021

Commande 4 / 20 mA [30 juin 2022]

```
void commandeVanne (const uint32_t inNumeroVanne, // 0 à 3  
                     const uint16_t inCommande) ;
```

La fonction `commandeVanne` présente deux arguments :

- le numéro (0 à 3) de la vanne que l'on veut commander (si l'argument est > 3, l'appel est ignoré)
- `inCommande`, la valeur image du courant de sortie (**attention sa valeur est maintenant sur 16 bits**).

La valeur de `inCommande` est calculée comme suit :

```
inCommande = uint16_t (courant_en_mA * 196.56)
```

Par exemple :

- pour 4 mA, la commande est 786 ;
- pour 20 mA, la commande est 3931.

Attention, l'électronique de commande est conçue pour assurer un courant proportionnel à la commande dans la plage [4 mA, 20 mA]. En dehors, des non linéarités peuvent apparaître.

Retour d'une information logique 0-24V

Les ports du micro-contrôleur utilisés pour les retours logiques sont :

```
static const uint8_t PORT_ENTREE_LOGIQUE_CAPTEUR_VANNE_0 = PG2 ;  
static const uint8_t PORT_ENTREE_LOGIQUE_CAPTEUR_VANNE_1 = PG3 ;  
static const uint8_t PORT_ENTREE_LOGIQUE_CAPTEUR_VANNE_2 = PG0 ;  
static const uint8_t PORT_ENTREE_LOGIQUE_CAPTEUR_VANNE_3 = PG9 ;
```

Pour les lire, deux possibilités.

Utiliser digitalRead. En argument, utiliser l'une des constantes ci-dessus. L'appel retourne :

- la valeur **LOW** (c'est-à-dire 0), si la tension est proche de 0V ;
- la valeur **HIGH** (c'est-à-dire 1), si la tension est proche de 24V.

Utiliser retourLogiqueVanne.

```
bool retourLogiqueVanne (const uint32_t inNumeroVanne) ;
```

L'appel retourne :

- la valeur **false**, si la tension est proche de 0V ;
- la valeur **true**, si la tension est proche de 24V.

La fonction renvoie toujours **false** si l'argument est > 3.

Si la vanne est programmée pour fournir une tension analogique, la valeur renournée par ces fonctions n'est pas significative.

Retour d'une information analogique 0-10V

```
uint16_t retourAnalogiqueVanne (const uint32_t inNumeroVanne) ;
```

La fonction **retourAnalogiqueVanne** retourne un entier qui reflète la tension renvoyée par la vanne. La valeur renvoyée est comprise entre 0 et 511.

La tension en Volt est obtenue à partir de la valeur renvoyée par :

```
tension_en_volt = valeur_renvoyée / 51.1
```

Par exemple : si la valeur renvoyée est 233, la tension est 4,56 V.

L'électronique est conçue pour renvoyer une valeur proportionnelle à la tension d'entrée dans la plage [0, 10V]. Au delà de 10V, la valeur renvoyée est toujours 511.

Si la vanne est programmée pour un retour logique, la valeur renvoyée est soit voisine de 0, soit 511.



**Point de test
d'une commande de vanne**

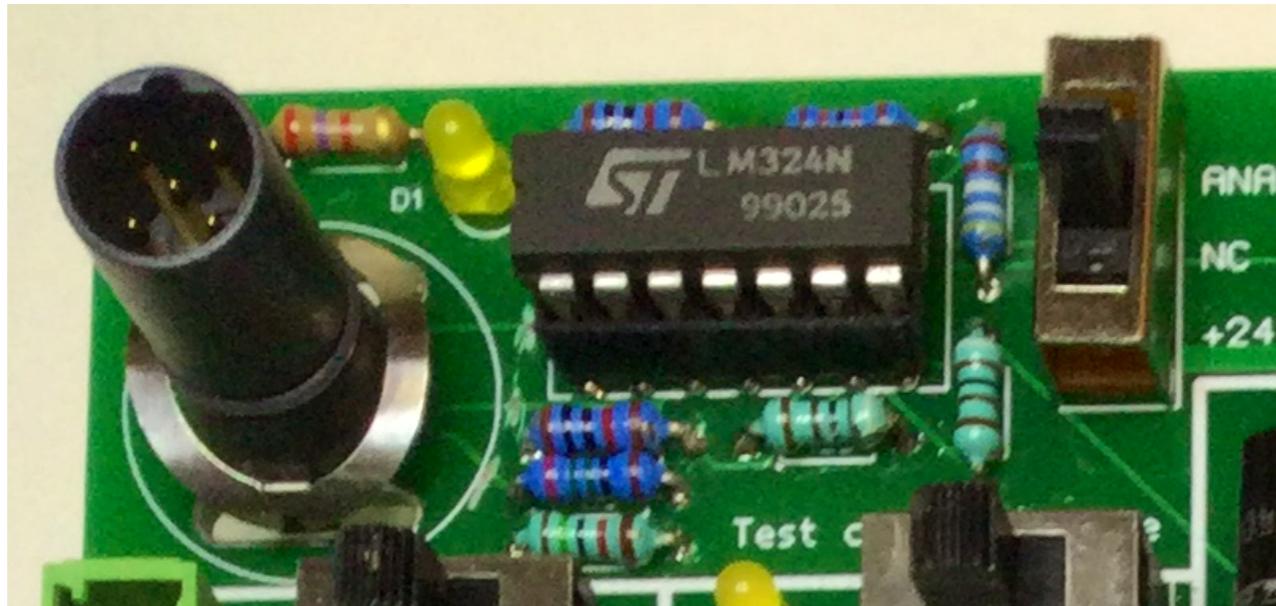
Résumé

Le point de test contrôle la connexion à une vanne :

- de vérifier la présence de la tension de +24V ;
- de mesurer le courant délivré par la sortie 4 / 20 mA ;
- de renvoyer une tension logique 0 - 24V ou une tension analogique 0 - 10V.

Il permet de vérifier le bon fonctionnement de l'électronique associée et les câbles de connexion à une vanne : un même câble sert pour se connecter à une vanne ou au point de test.

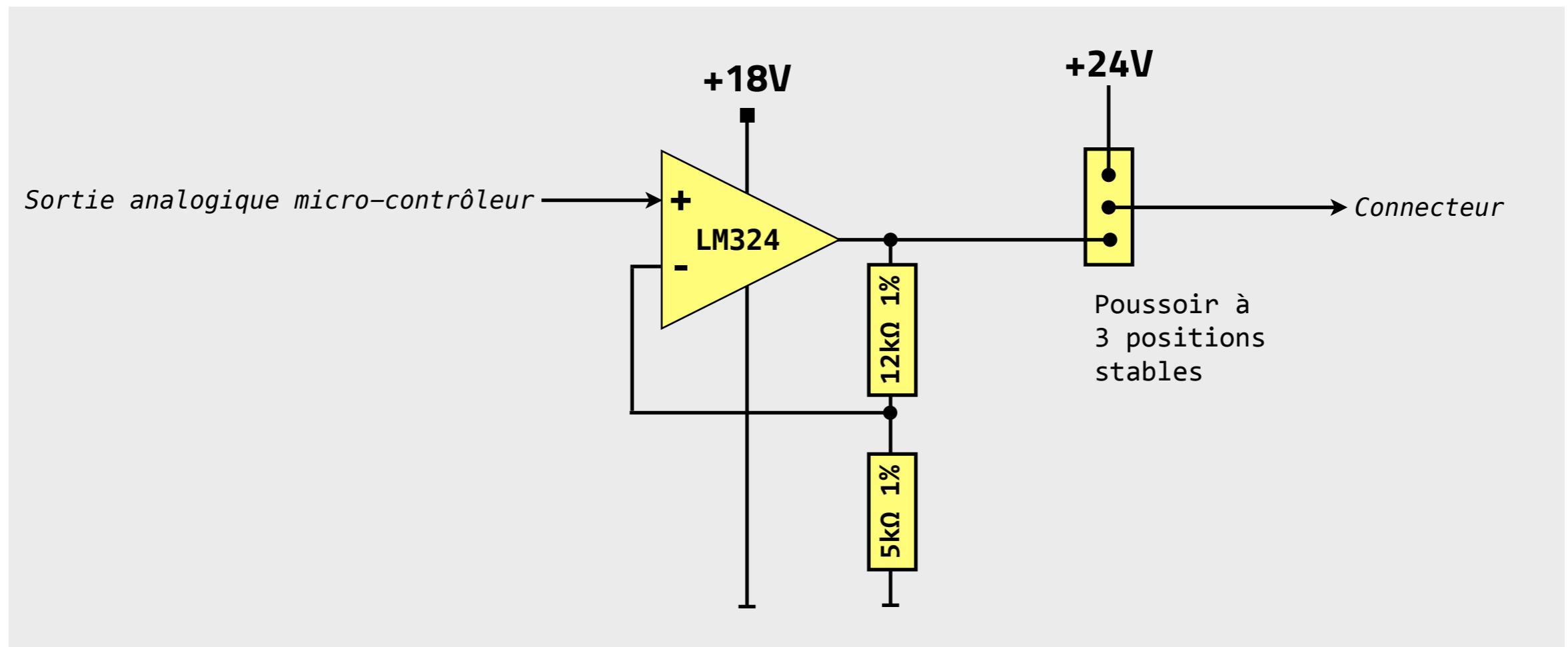
Le point de test



Le point de test comprend les interfaces suivantes :

- le connecteur M12 ;
- la led visualisant la présence de la tension de +24V ;
- l'interrupteur à glissière à 3 positions paramétrant le retour :
 - ANALOGIQUE : le retour est une tension analogique 0 - 10V ;
 - NC : le signal de retour n'est pas connecté, il est vu par l'électronique de contrôle d'une vanne comme une tension proche de 0V, c'est-à-dire un niveau logique 0 ;
 - +24V : le signal de retour est la tension de 24V, il est vu par l'électronique de contrôle d'une vanne comme un niveau logique 1.

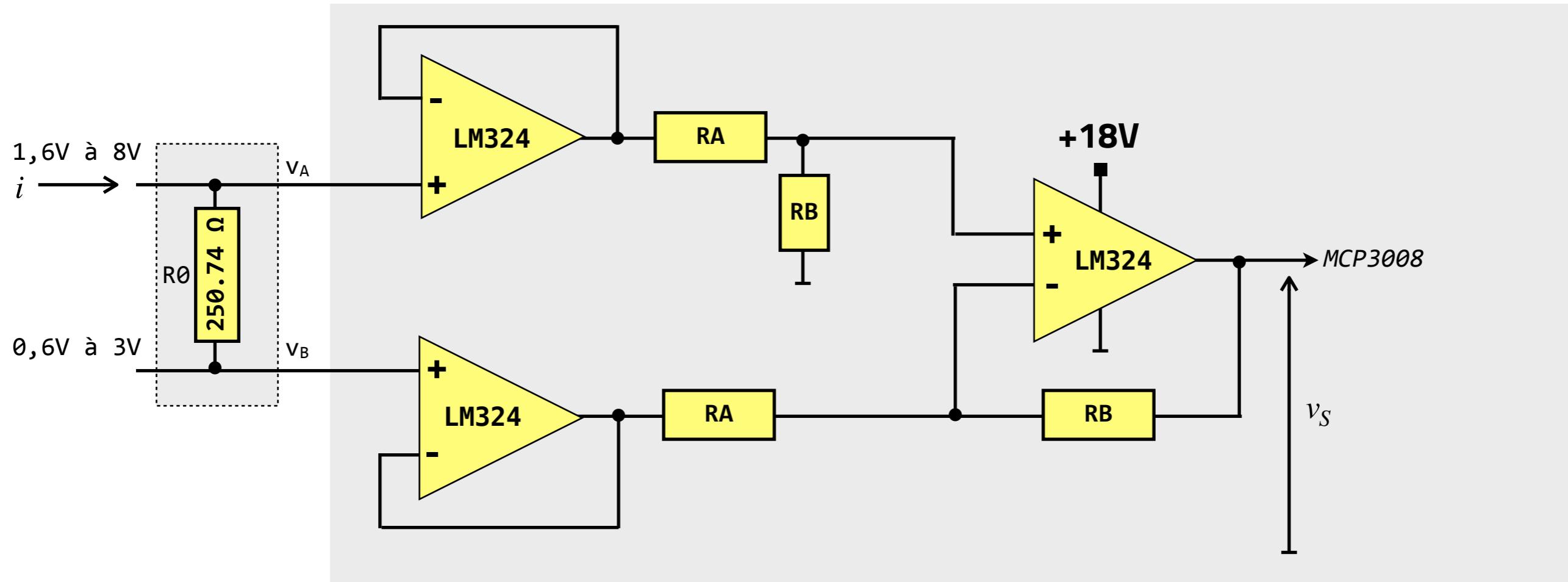
Schéma de la connexion du retour vers la vanne



La sortie du micro-contrôleur est un DAC 8 bits délivrant 3,3V à pleine échelle. Pour une valeur N ($0 \leq N \leq 255$), la tension de sortie v_s est :

$$v_s = \frac{5 \text{ k}\Omega + 12 \text{ k}\Omega}{5 \text{ k}\Omega} * \frac{3.3 \text{ V}}{255} * N = N * 0.044$$

Conversion 4 / 20 mA -> tension



La résistance d'entrée R0 est constituée d'une résistance de 255Ω en parallèle avec une résistance de $15k\Omega$:

$$R0 = \frac{255 \Omega * 15 k\Omega}{255 \Omega + 15 k\Omega} = 250.74 \Omega$$

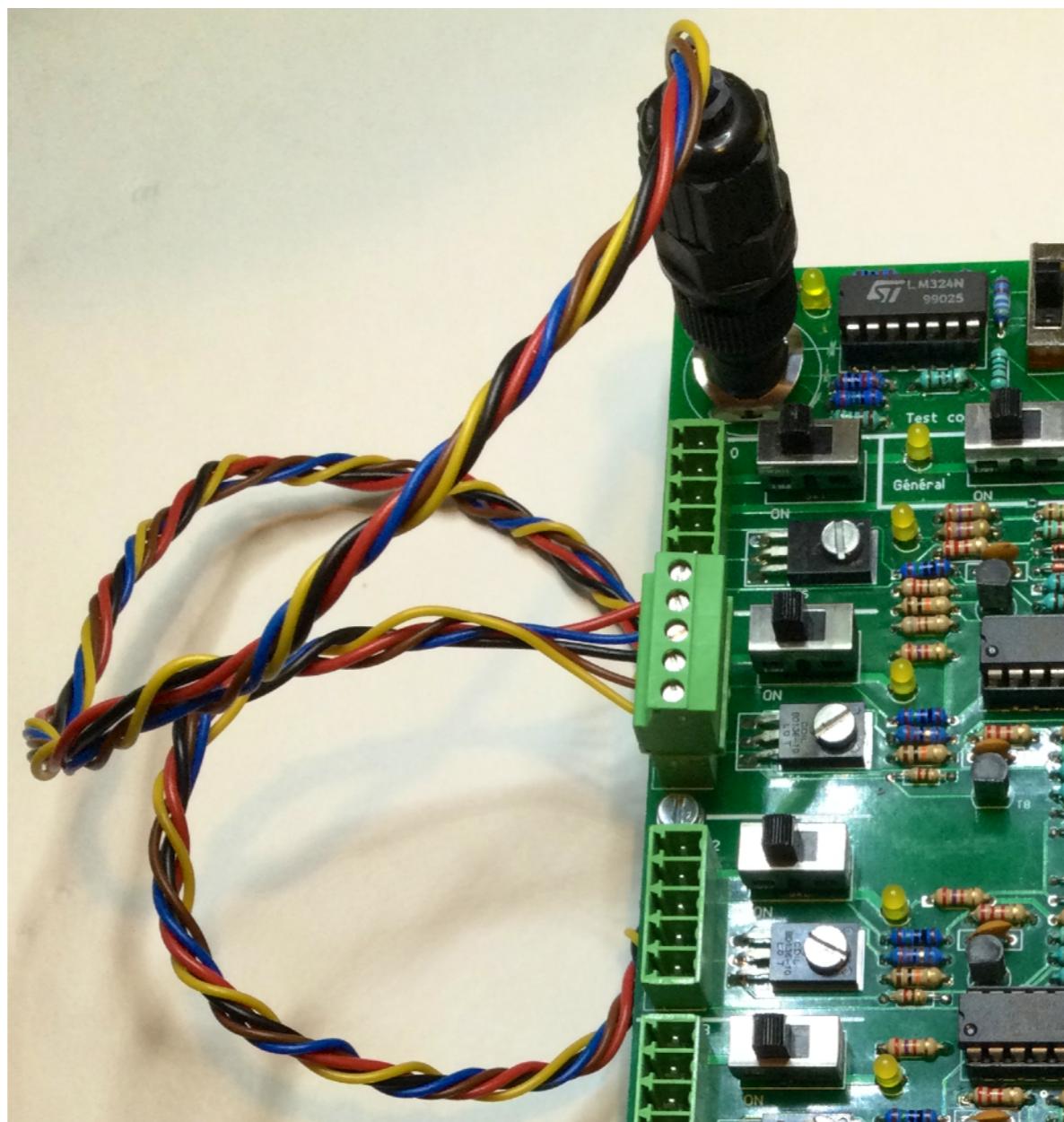
On choisit $RA=10 k\Omega$ et $RB=9,1 k\Omega$. La tension de sortie du montage est :

$$v_s = \frac{RB}{RA} * (v_A - v_B) = \frac{9.1 k\Omega}{10 k\Omega} * 250.74 \Omega * i$$

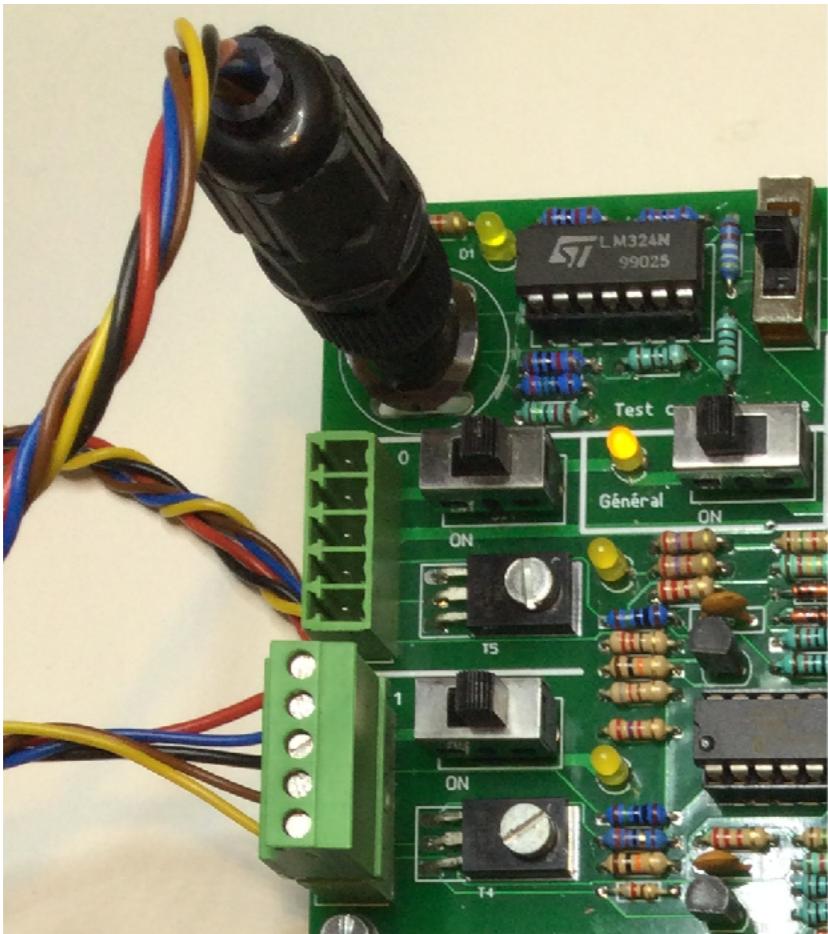
Le MCP3008 est un convertisseur 9 bits alimenté sous 5V :

$$valeur_retour = \frac{511}{5V} * v_s = \frac{511}{5V} * \frac{9.1 k\Omega}{10 k\Omega} * 250.74 \Omega * \frac{i (mA)}{1000 mA/A} = i (mA) * 23.3119$$

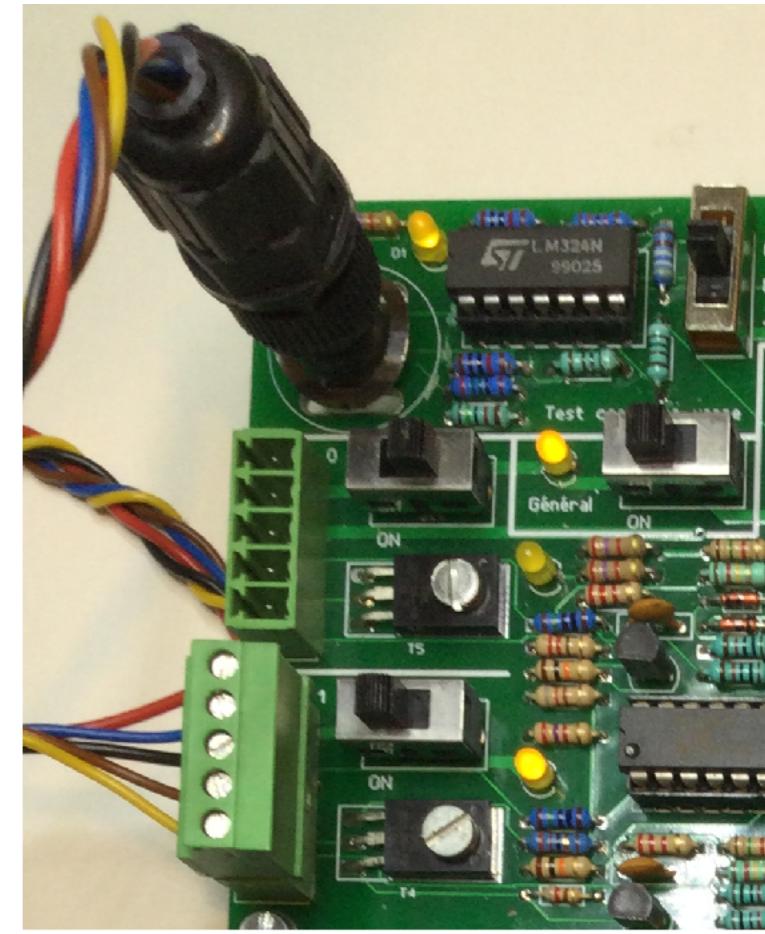
Connexion du point de test à une commande de vanne



Test de l'alimentation +24V



L'interrupteur de puissance de la vanne 1 ou
l'interrupteur général est OFF.



L'interrupteur de puissance de la vanne 1 et
l'interrupteur général sont ON.

Test de la commande 4 / 20 mA

```
uint16_t testCommandeVanne (void) ;
```

La fonction **testCommandeVanne** retourne un entier qui reflète la valeur du courant envoyé à la vanne. La valeur renvoyée est comprise entre 0 et 511.

Le courant mesuré en mA est obtenu à partir de la valeur renvoyée par :

$$\text{courant_en_mA} = \text{valeur_renvoyée} * 0,042883$$

Par exemple : si la valeur renvoyée est 464, le courant est 19,89mA.

Test du retour analogique 0 - 10V

```
void fixerValeurTestRetourAnalogique (const uint16_t inValeur) ;
```

La fonction **fixerValeurTestRetourAnalogique** fixe la tension du retour analogique vers la vanne si l'interrupteur à glissière est sur la position ANALOGIQUE. Si l'interrupteur à glissière est sur une autre position, la tension délivrée par l'électronique est ignorée.

Le tension renvoyée en Volt est obtenue à partir de **inValeur** par :

tension_en_volt = inValeur * 0,04406

Par exemple : si **inValeur** est 195, la tension renvoyée est 8,59V.

Note :

- la valeur maximum renvoyée est 11,24V, ce qui correspond à **inValeur** égal à 255 ;
- si **inValeur** est supérieur à 255, la valeur effectivement utilisée est 255.



Croquis d'exemple

Les croquis d'exemple

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

- `01-afficheur-lcd` ;
- `02-ne-pas-utiliser-delay` ;
- `03-croquis-test-es-utilisateur` ;
- `04-croquis-test-entrees-logiques` ;
- `05-croquis-test-entrees-analogiques` ;
- `06-croquis-test-commande-vannes`.

Note. pensez à effectuer l'opération précédemment décrite :

- ponts de soudure marqués **SB14** et **SB15** au dos de la carte **NUCLEO-H743ZI2** ([voir à cette page](#)).

Composition des croquis

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

- `STM32H743-configuration-1s2n.cpp`
- `STM32H743-configuration-1s2n.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).

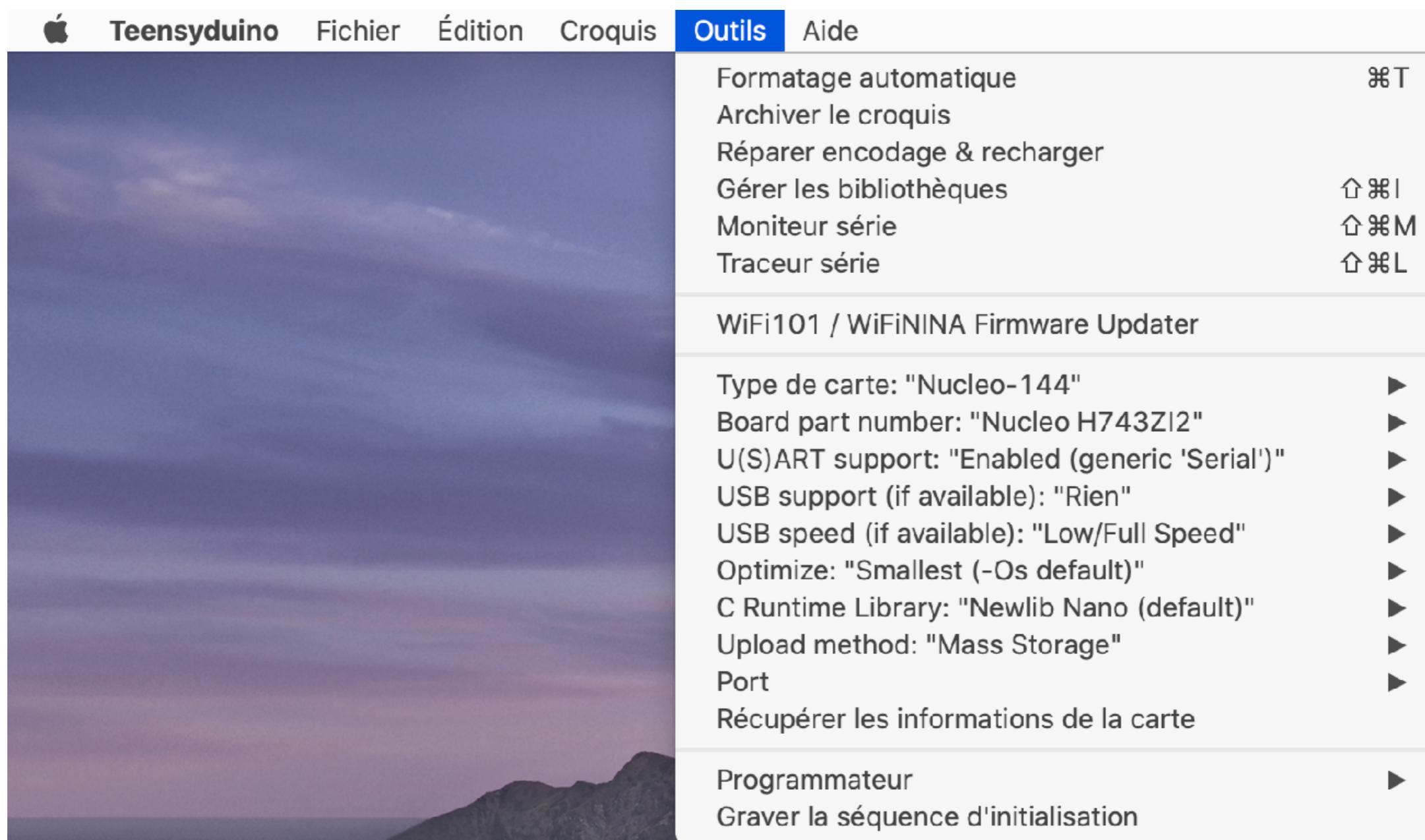
Dans la fonction **setup** du croquis, appeler la fonction **configurerCarteH743LS2N**.

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-1s2n.h` au début du fichier :

```
#include "STM32H743-configuration-1s2n.h"
```

- ② Dans la fonction `setup`, appeler une et une seule fois la fonction `configurerCarteH743LS2N` :

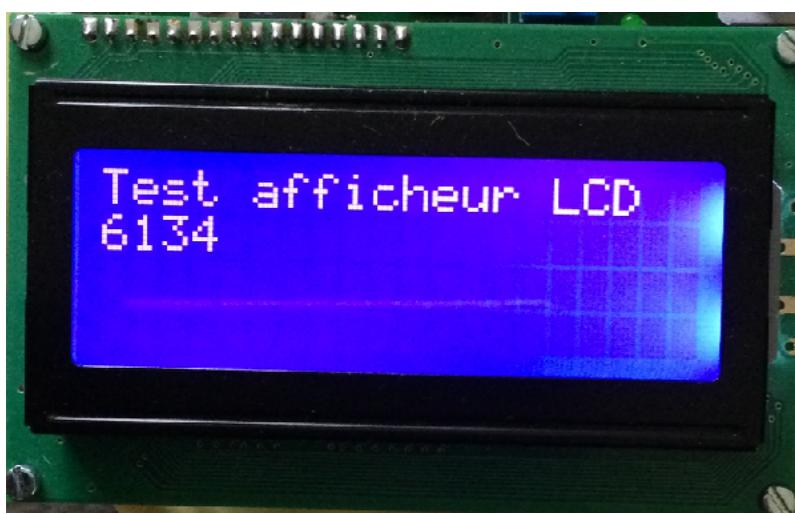
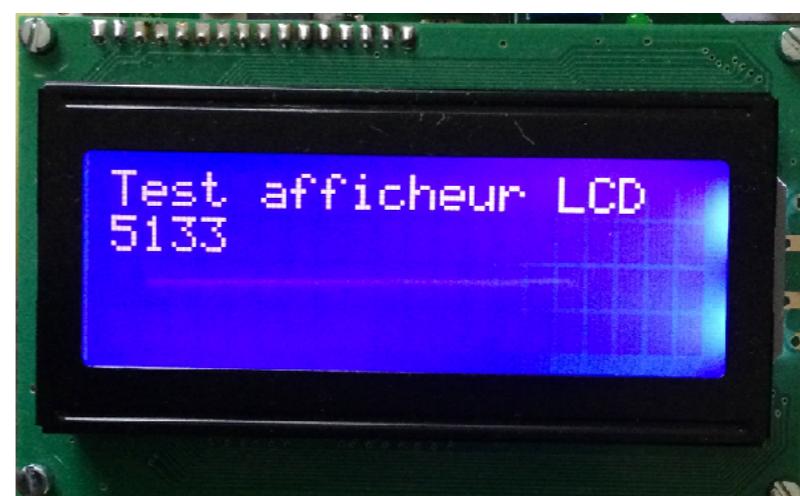
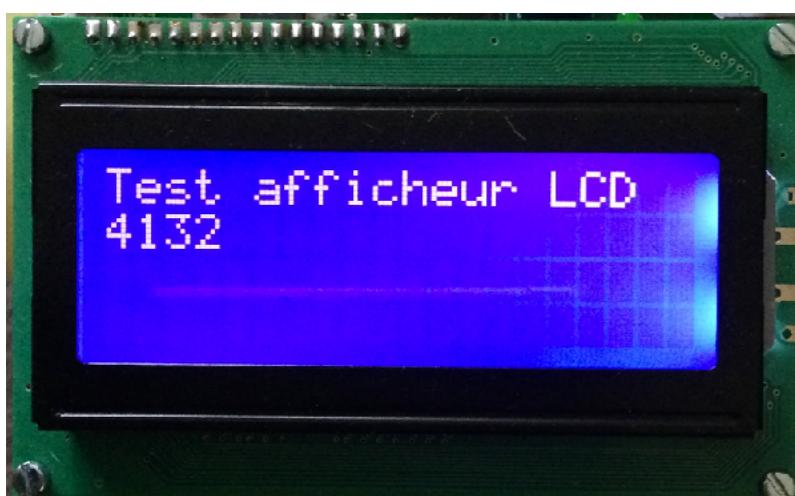
```
void setup () {  
    configurerCarteH743LS2N () ;  
    ...  
}
```

- ③ Ensuite, toutes les fonctions citées dans `STM32H743-configuration-1s2n.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.

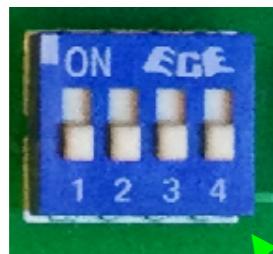
Le croquis définit 4 actions périodiques :

- 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-utilisateur (1/2)

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



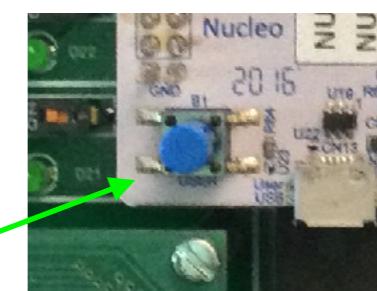
Interrupteur DIL : OFF -> 1, ON -> 0



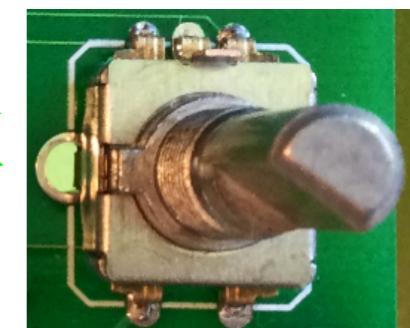
Poussoirs : relâché -> 1, appuyé -> 0



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 *borneInf* \leq *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.

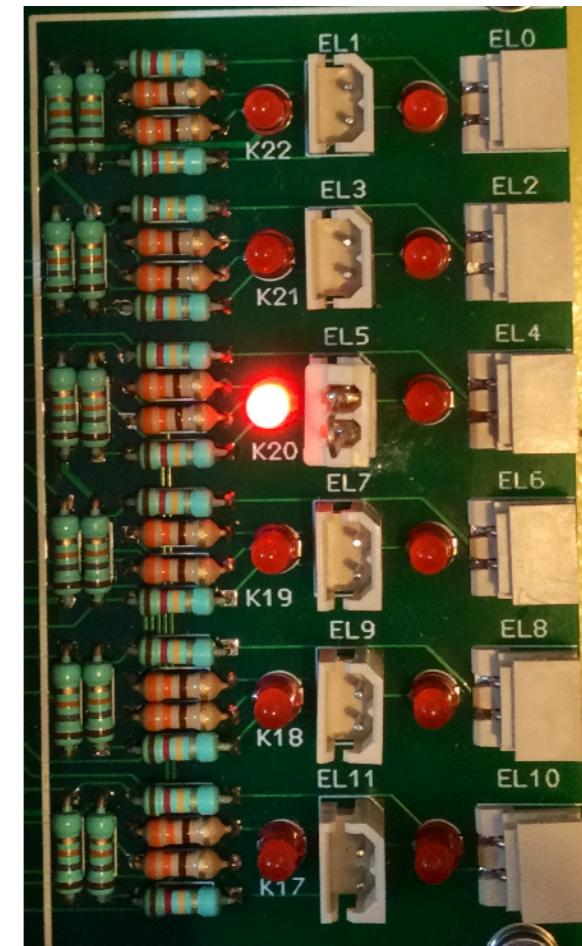
Croquis 04-croquis-test-entrees-logiques

Ce croquis teste les 12 entrées logiques EL0 à EL11.

L'afficheur affiche en permanence la valeur des entrées logiques :

- le chiffre le plus à gauche correspond à EL11 ;
- le chiffre le plus à droite correspond à EL0 ;
- un chiffre à 1 correspond à une entrée non connectée (led éteinte) ;
- un chiffre à 0 correspond à une entrée activée (led allumée).

Affichage, lorsque EL5 est seul activée.



Croquis 05-croquis-test-entrees-analogiques

Ce croquis teste les 16 entrées analogiques EA0 à EA15.

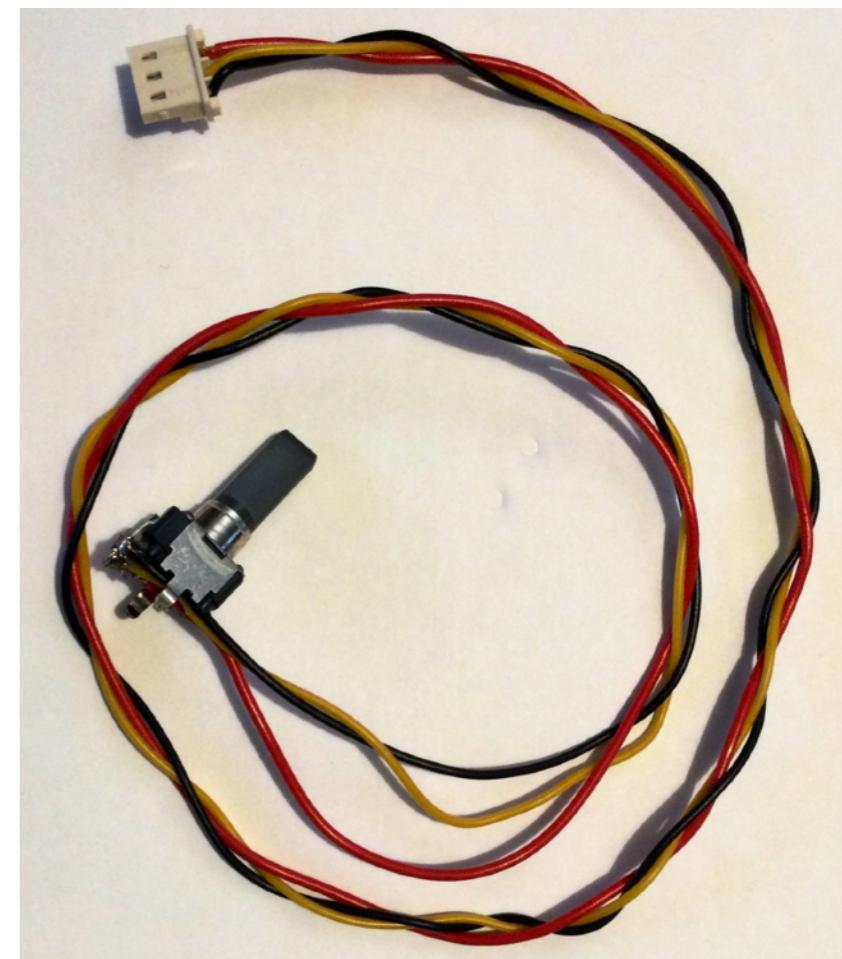
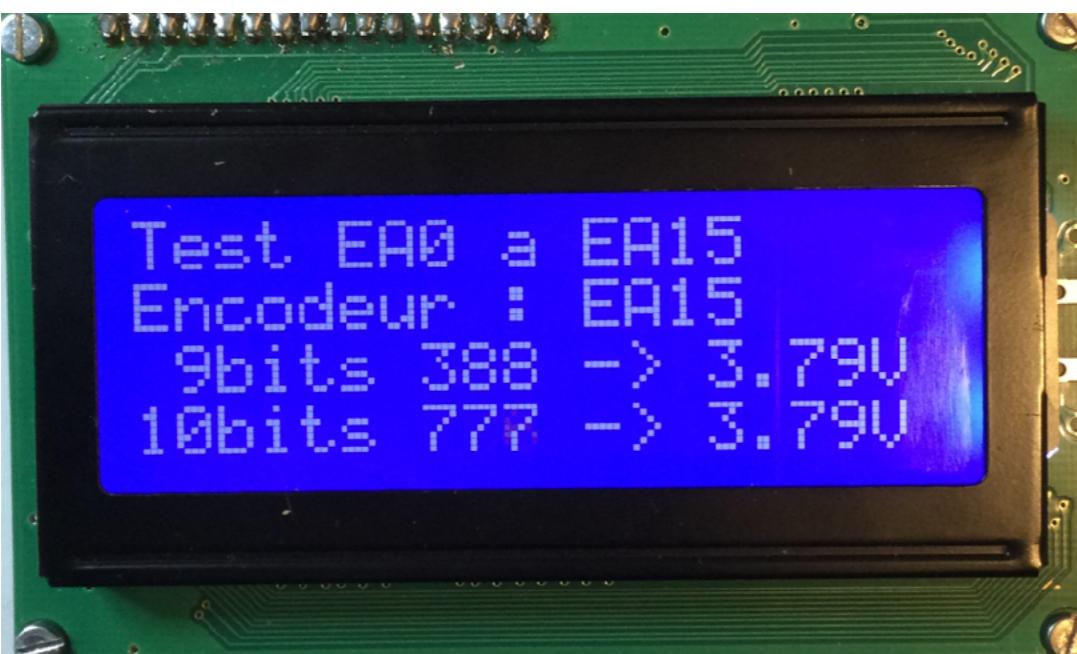
L'encodeur numérique permet de sélectionner l'entrée analogique.

Pour fixer une tension analogique sur EA0 à EA14, utiliser le bouchon équipé d'un potentiomètre.

Il y a deux fonctions possibles pour lire une entrée analogique :

- `lireEntreeAnalogique_9bits`, qui renvoie une valeur entre 0 et 511 (3^e ligne) ;
- `lireEntreeAnalogique_10bits`, qui renvoie une valeur entre 0 et 1023 (4^e ligne).

Exemple d'affichage de lecture de l'entrée EA15 (potentiomètre sur la carte).

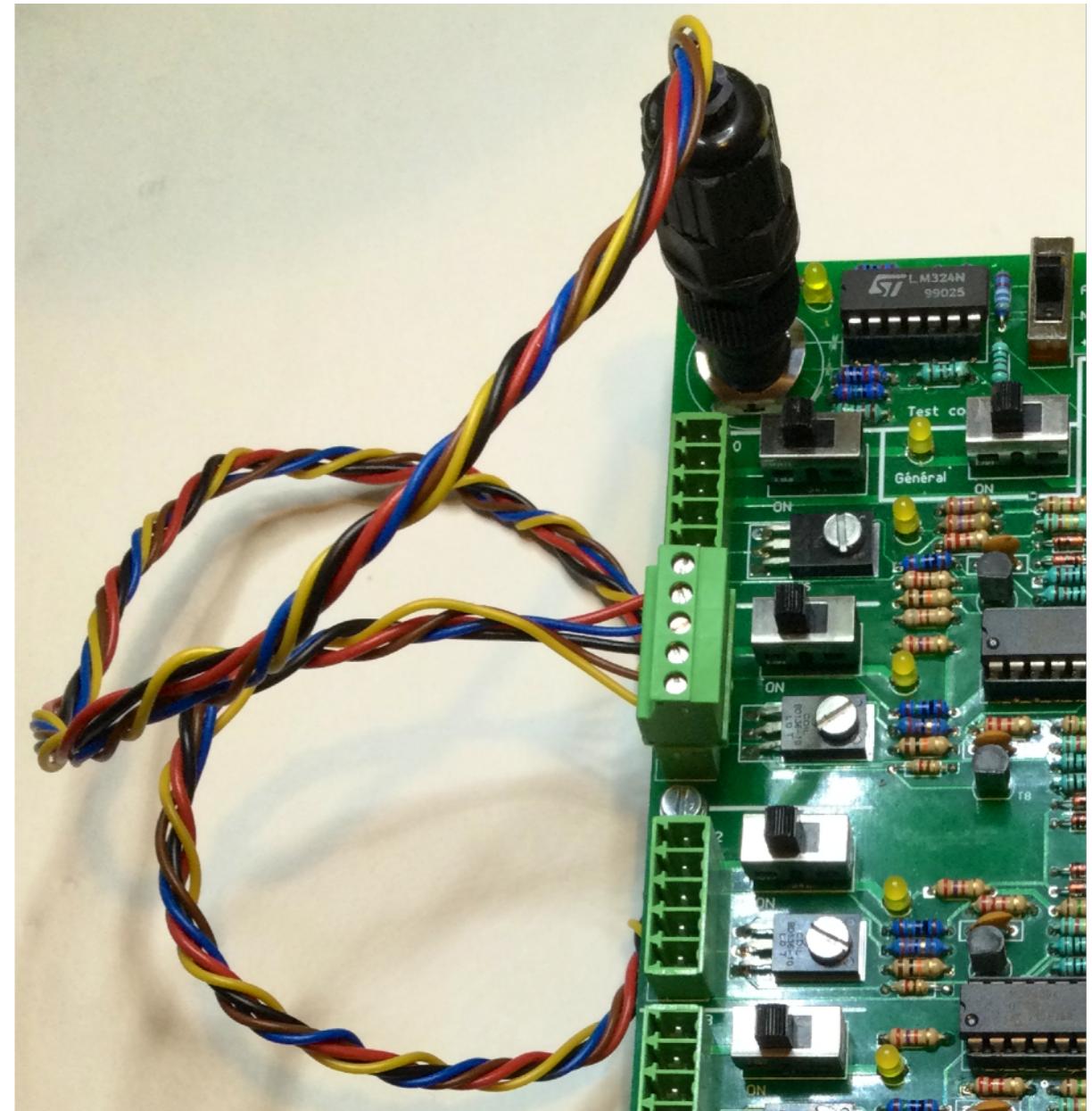


Croquis 06-croquis-test-commande-vannes (1/4)

Ce croquis permet de tester un câble de connexion vanne et / ou l'électronique associée :

- l'encodeur numérique permet de sélectionner la sortie vanne (0 à 3) ;
- l'interrupteur DIL4 sélectionne l'affichage : commande 4 / 20 mA ou retour analogique.

Pour tester un câble de connexion vanne et / ou l'électronique associée, il faut relier la sortie testée au point de test.



Croquis 06-croquis-test-commande-vannes (2/4)

Test du retour logique

Le croquis reporte le retour logique vers la led LED-0. Donc, en fonction de la position de l'interrupteur à glissière à 3 positions :

- +24V , le retour est 24V : la led LED-0 est allumée ;
- NC, le retour est une tension proche de 0V : la led LED-0 est éteinte ;
- ANALOG, le retour est une tension inférieure à 11,24V, la led LED-0 est éteinte.

Croquis 06-croquis-test-commande-vannes (3/4)

Test commande 4 / 20 mA

L'interrupteur DIL4 doit être à OFF.

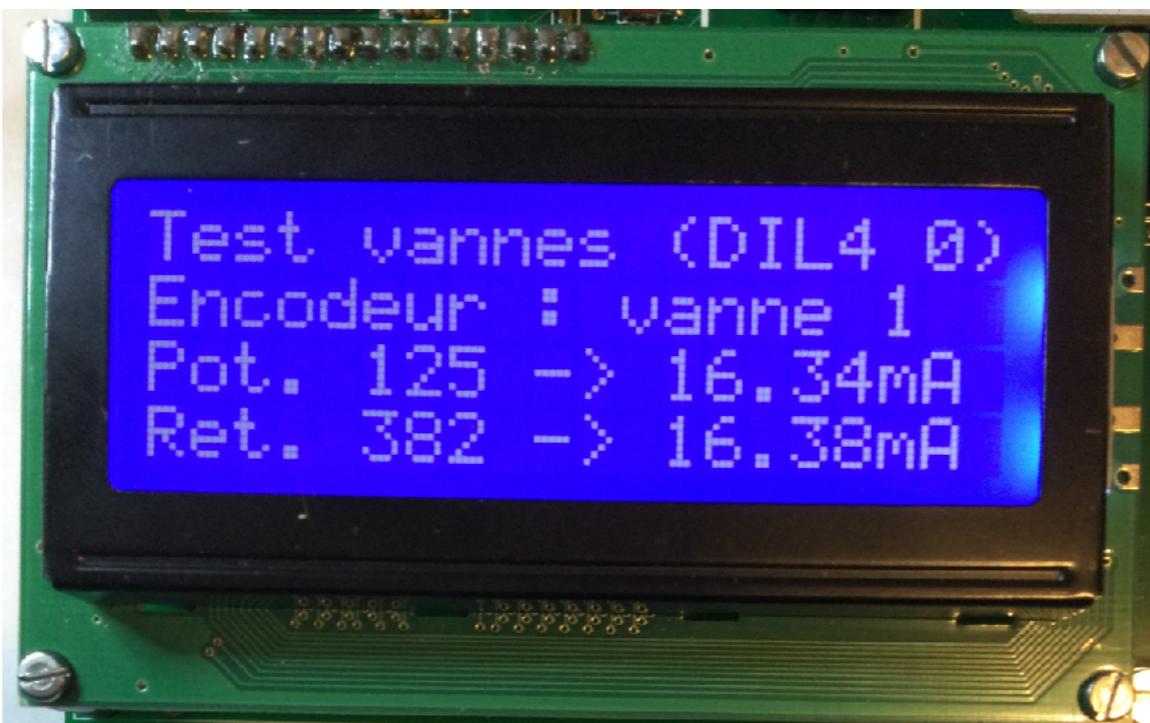
Le potentiomètre EA15 fixe la commande de la sorte 4 / 20 mA (par appel de la fonction commandeVanne). Sont affichés sur la 3^e ligne :

- la valeur entre 0 et 255 du potentiomètre EA15 ;
- et la valeur en mA correspondante.

Sont affichés sur la 4^e ligne :

- la valeur entre 0 et 511 renvoyé par la fonction testCommandeVanne ;
- et la valeur en mA correspondante.

En théorie, les deux valeurs en mA doivent être égales.



Croquis 06-croquis-test-commande-vannes (4/4)

Test retour analogique

L'interrupteur DIL4 doit être à ON.

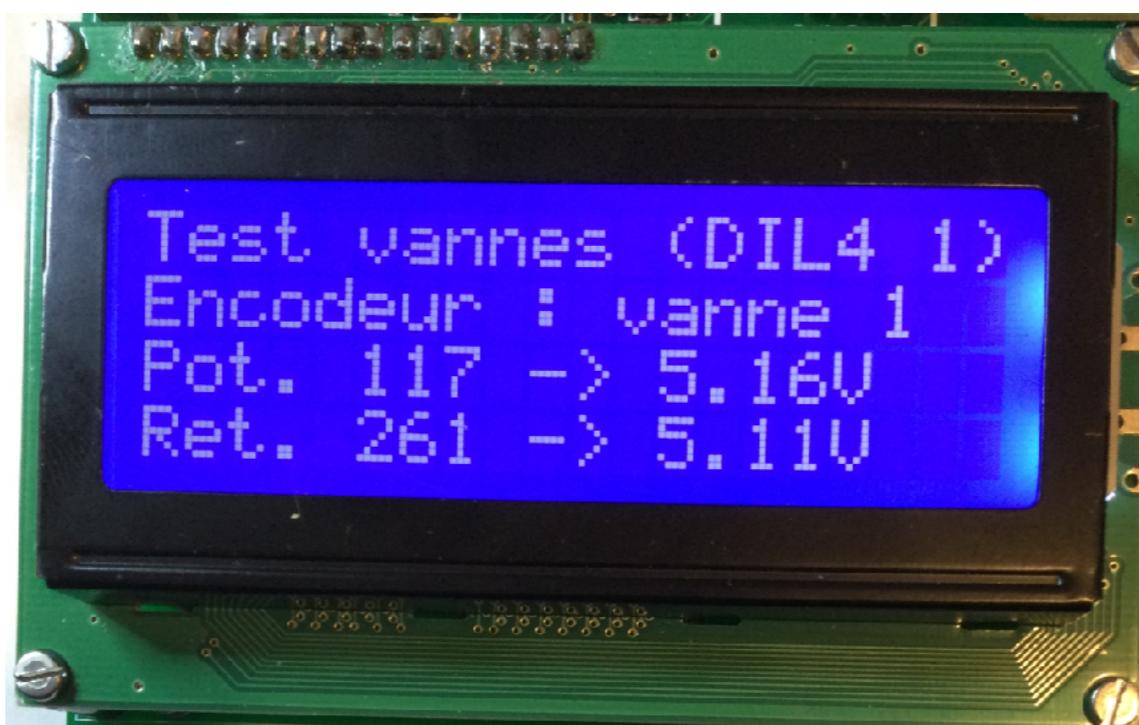
Le potentiomètre EA15 fixe la tension analogique de retour (par appel de la fonction `fixerValeurTestRetourAnalogique`). Sont affichés sur la 3^e ligne :

- la valeur entre 0 et 255 du potentiomètre EA15 ;
- et la valeur en Volt correspondante.

Sont affichés sur la 4^e ligne :

- la valeur entre 0 et 511 renvoyé par la fonction `retourAnalogiqueVanne` ;
- et la valeur en Volt correspondante.

En théorie, les deux valeurs en Volt doivent être égales.





Plan de la carte

Erreurs de conception de la carte

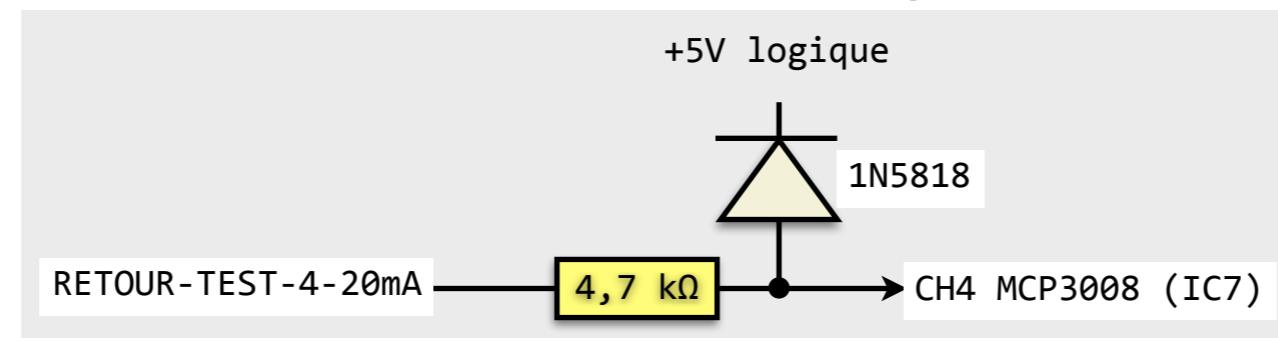
La carte comporte un certain nombre d'erreurs de conception.

Erreur 1. L'alimentation du LM7812 (IC15, page 1 du schéma) est directement le +24V. Comme ce régulateur doit pouvoir fournir 400 mA, la puissance qu'il dissipe est $(24V - 12V) * 0,4 A = 4,8W$. C'est trop ! La correction est d'interposer 3 résistances de $8,2\Omega$ 5W en série. Chaque résistance dissipe $8,2\Omega * (0,4A)^2 = 1,32W$. La puissance dissipée dans le régulateur est $(24V - 3 * 8,2\Omega * 0,4A - 12V) * 0,4 A = 0,87W$.

Erreur 2. Oubli des signaux de retour logique des vannes RETOUR-LOGIQUE-VANNE-*i*. Les connexions sont ajoutées (page 6 du schéma).

Erreur 3. Le potentiomètre EA15 (P26, page 10 du schéma) est monté à l'envers, le sens de rotation est inversé. Pas de correction.

Erreur 4. Pas de protection de surtension de l'entrée CH4 de IC7 (page 17 du schéma).



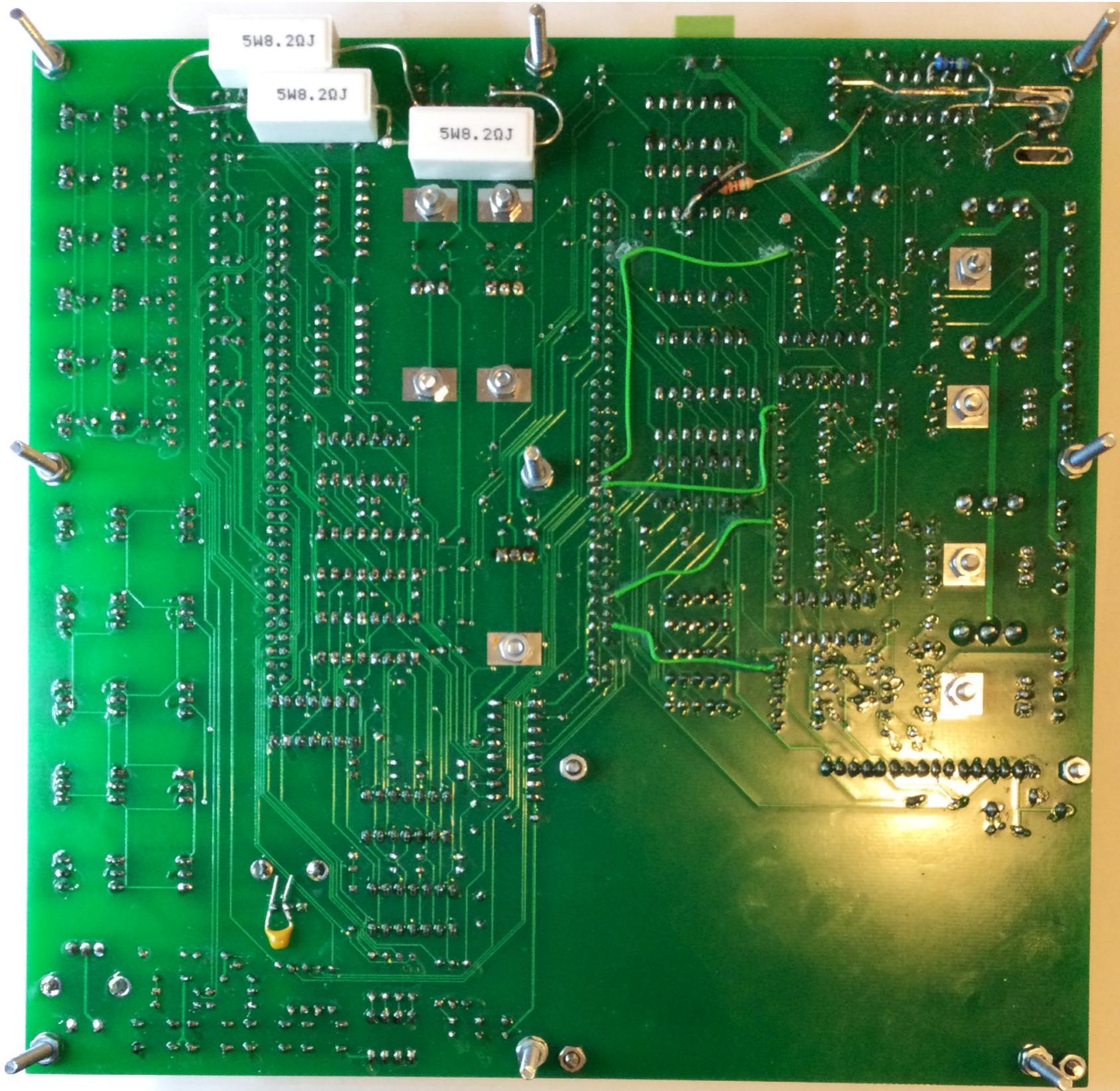
Erreur 5. Les signaux 2 et 5 du connecteur K5 (page 18 du schéma) doivent être échangés. Les pistes ont été coupés et des straps ont été ajoutés.

Erreur 6. Quand le point de test est déconnecté, la valeur renvoyée par `testCommandeVanne` est nulle. Une résistance $1M\Omega$ a été ajoutée entre les pattes 3 et 11 de l'IC1.

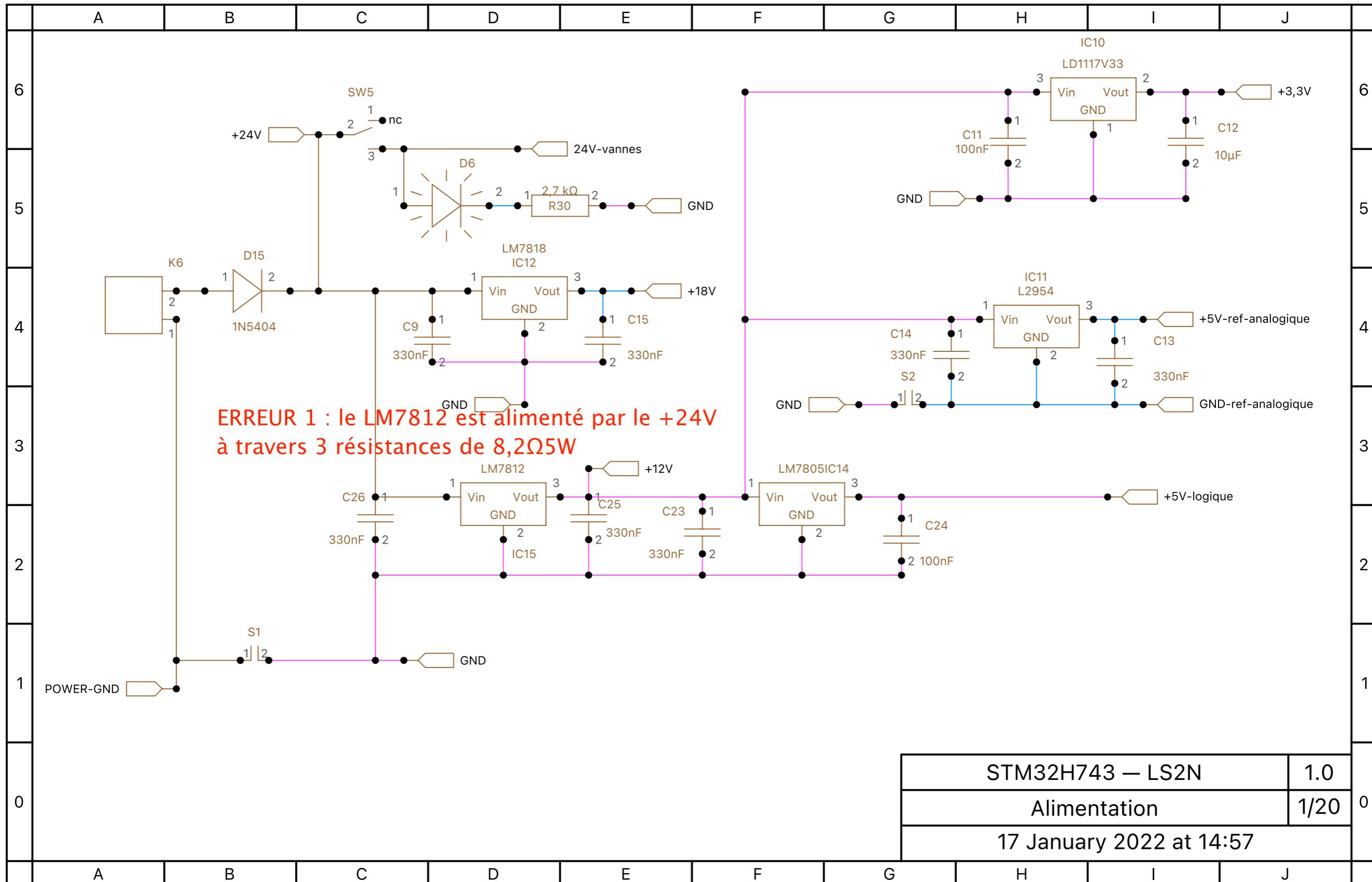
Choix de PA15 pour SPI3_CS

30 juin 2022. Le choix de PA15 pour être SPI3_CS n'est pas judicieux. En effet, avec la version 2.3.0 de STMDuino, ce signal n'est pas géré par le module SPI3. Il faut donc le gérer à la main.

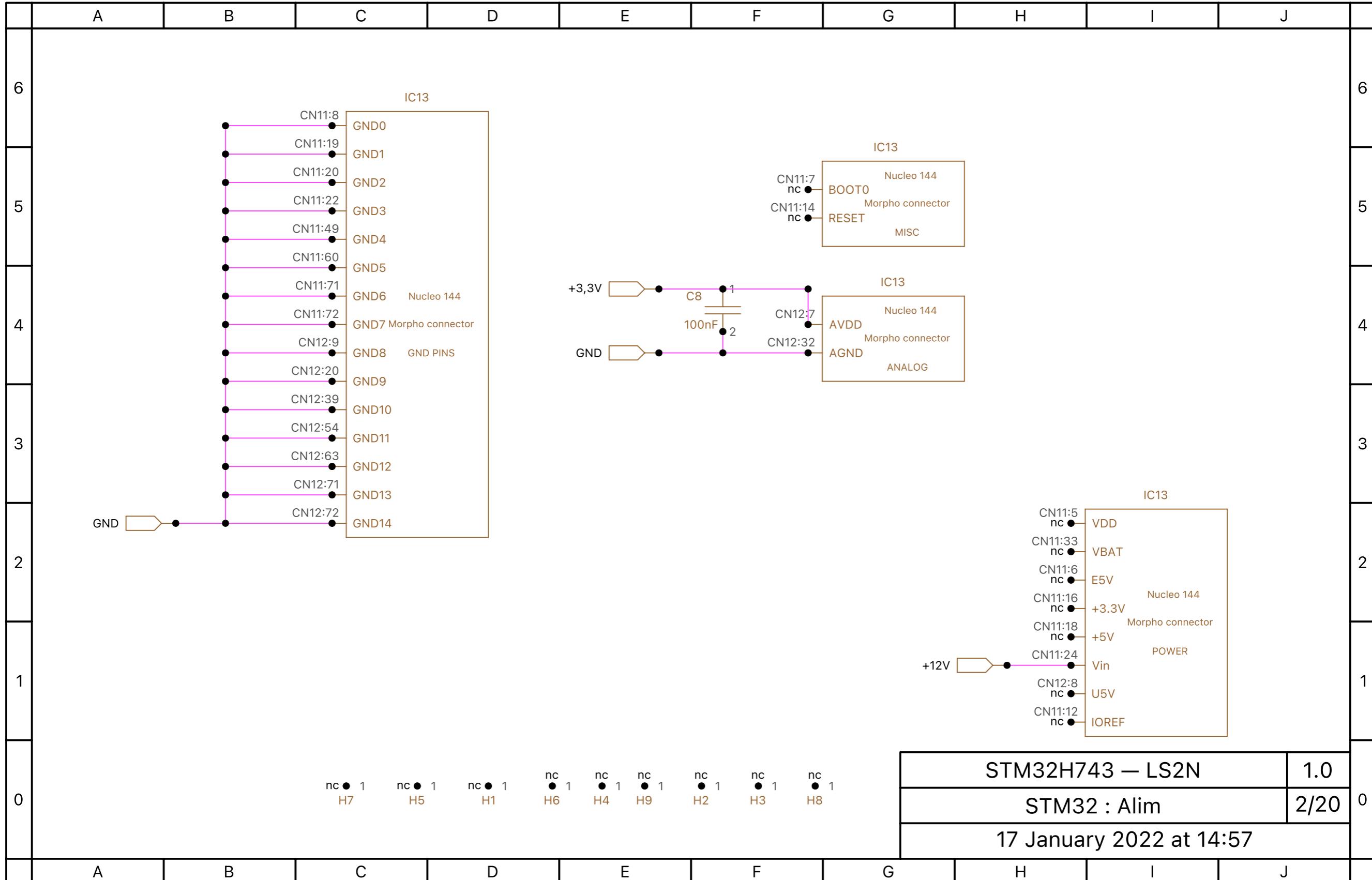
Vue de dessous



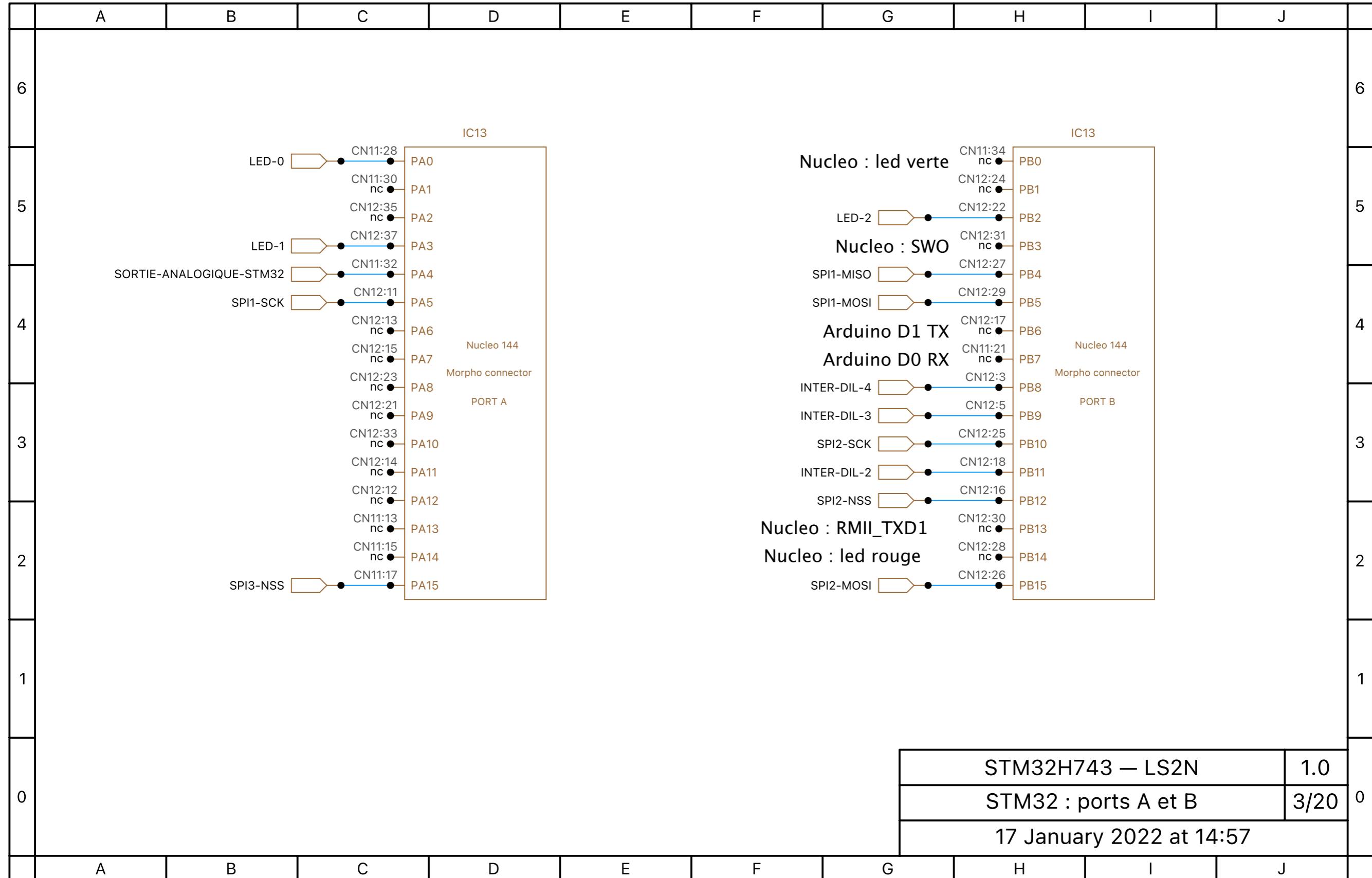
Plan de la carte (1/20)



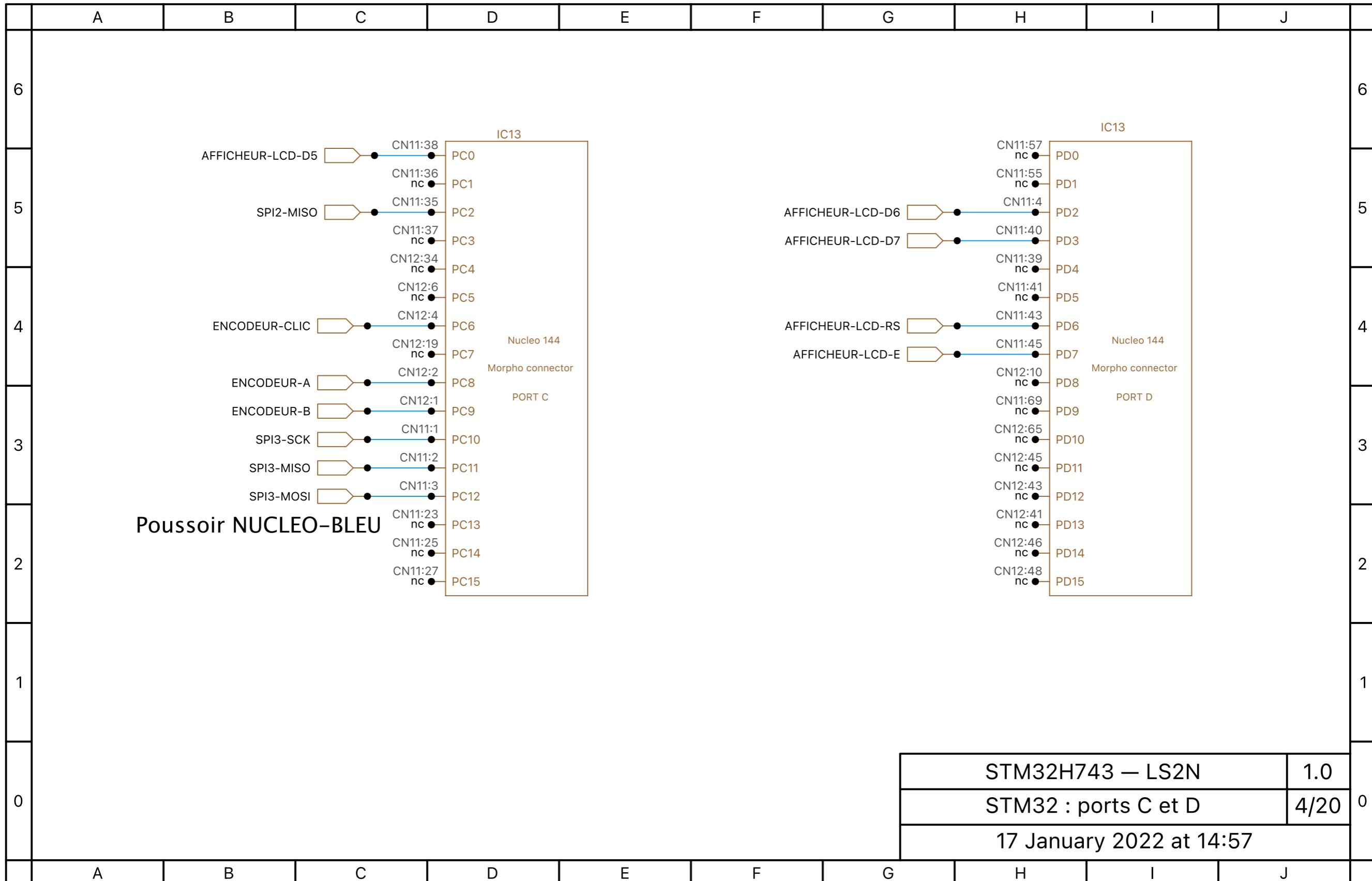
Plan de la carte (2/20)



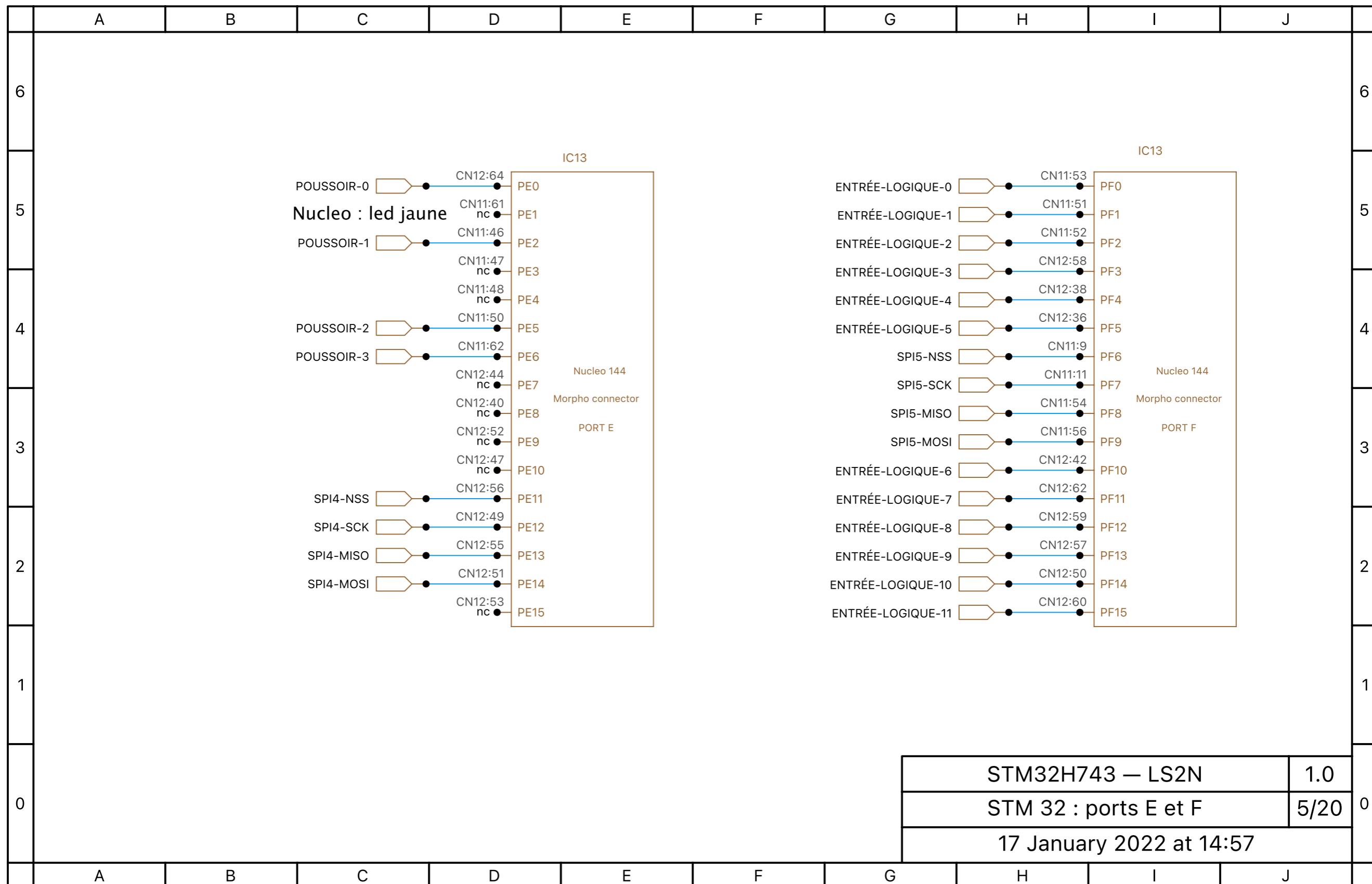
Plan de la carte (3/20)



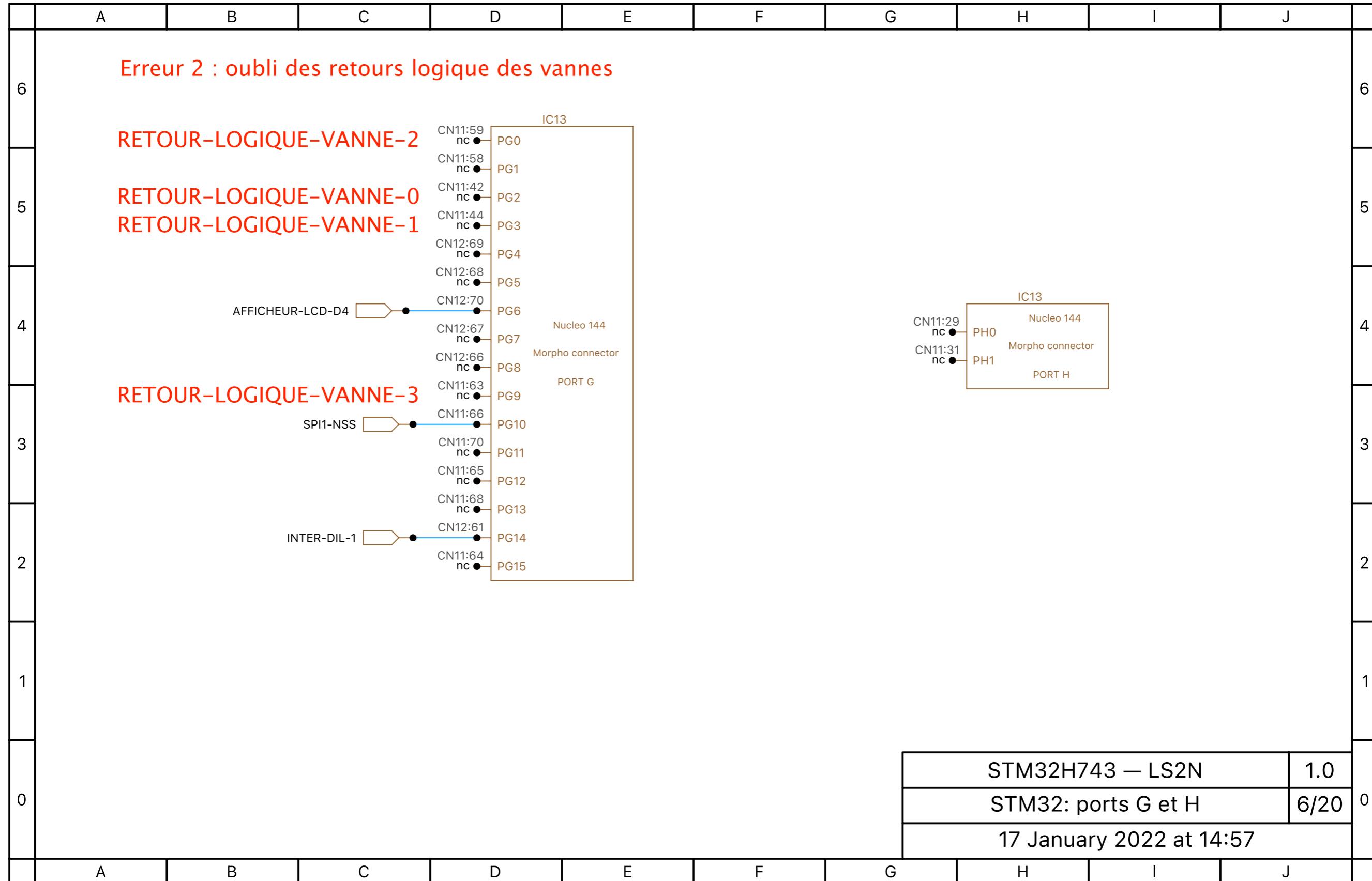
Plan de la carte (4/20)



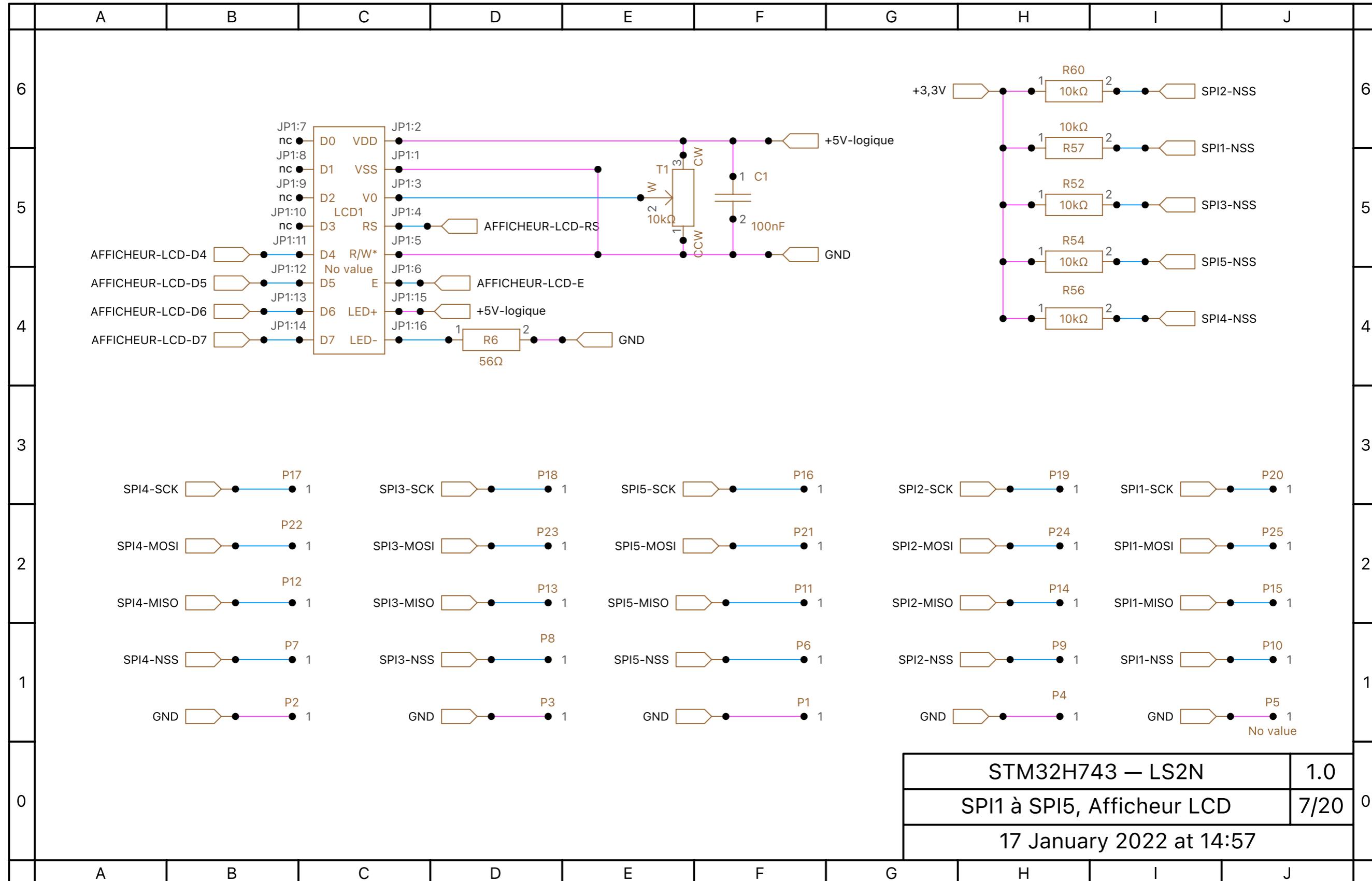
Plan de la carte (5/20)



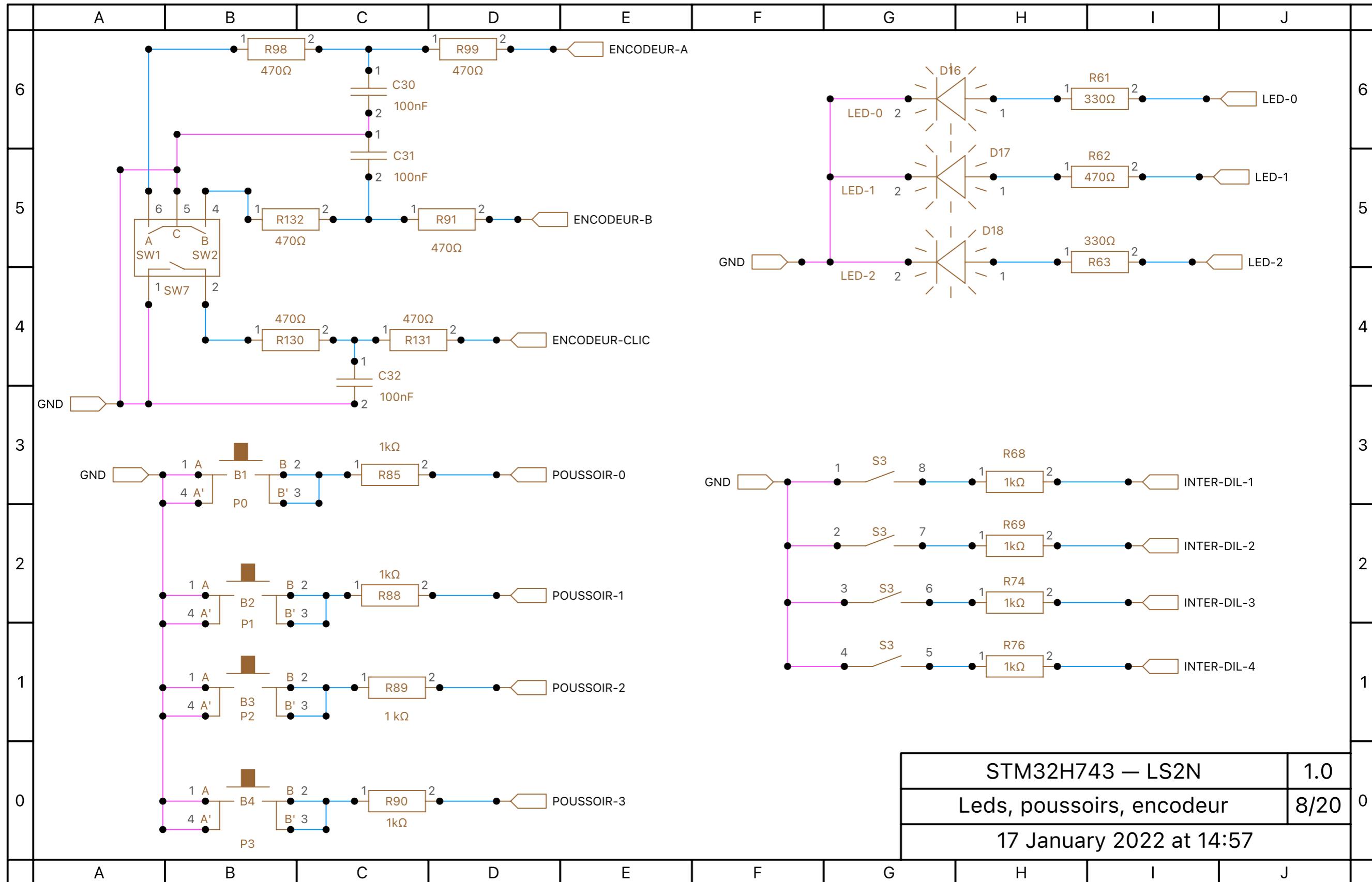
Plan de la carte (6/20)



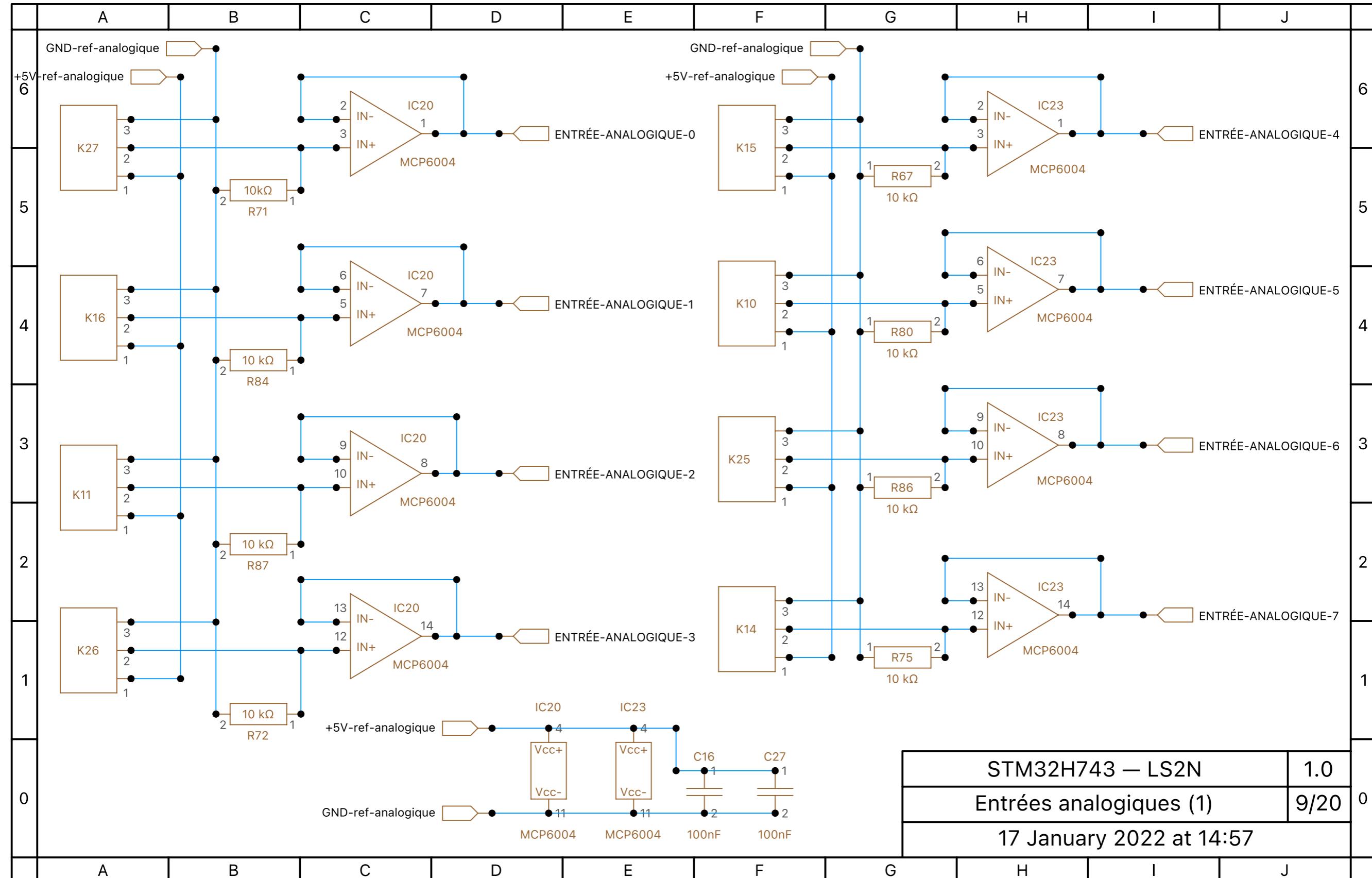
Plan de la carte (7/20)



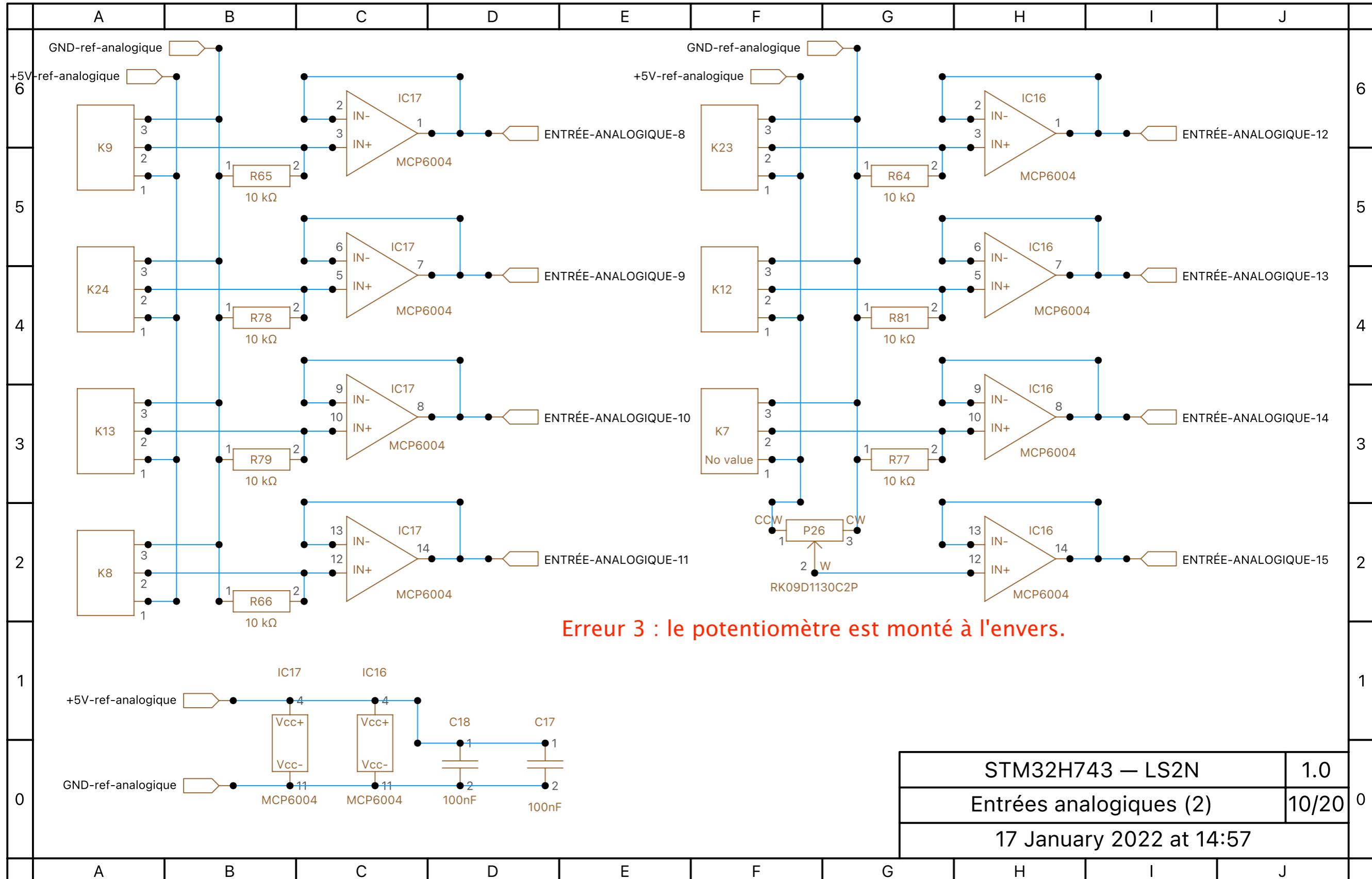
Plan de la carte (8/20)



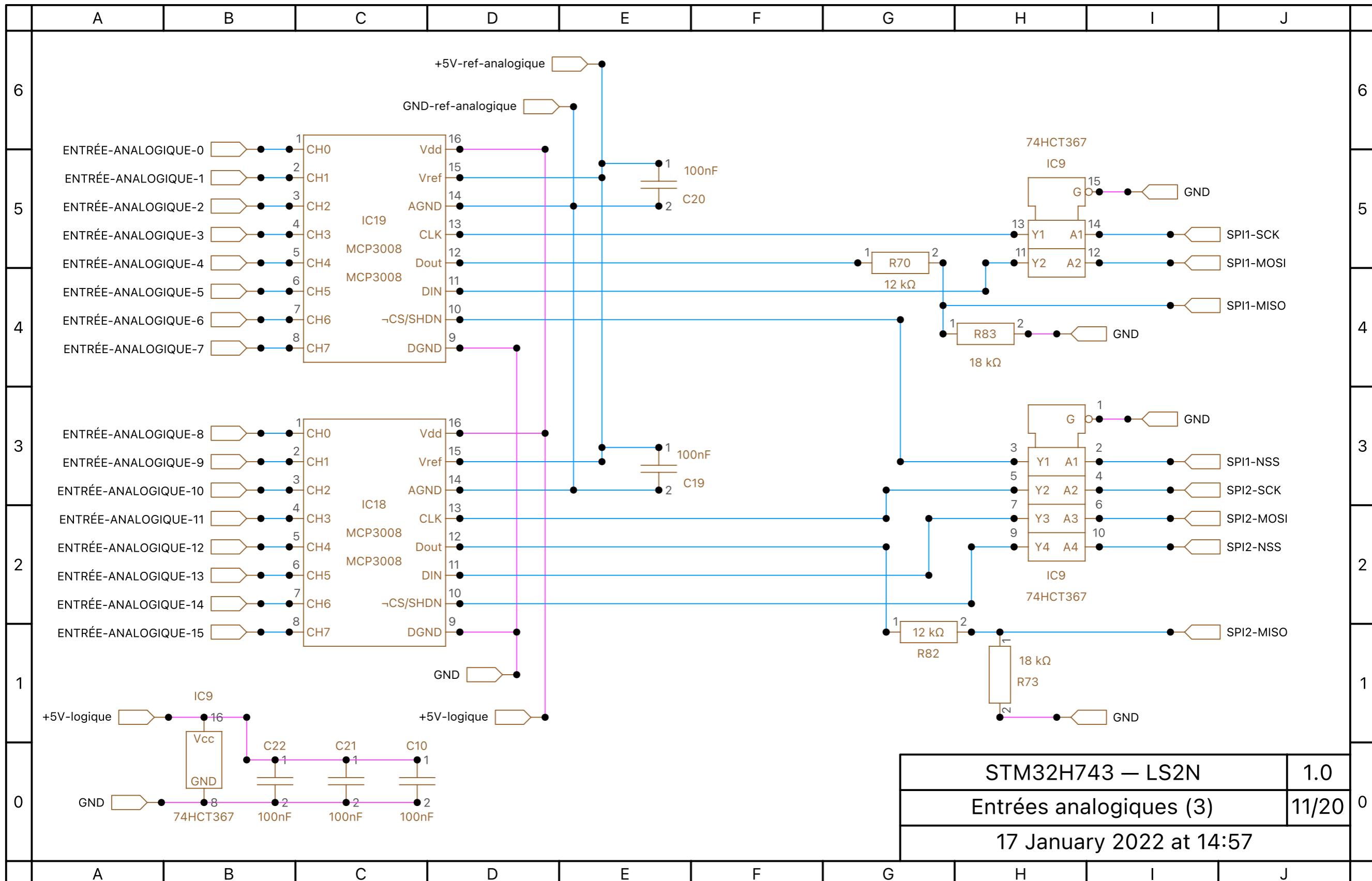
Plan de la carte (9/20)



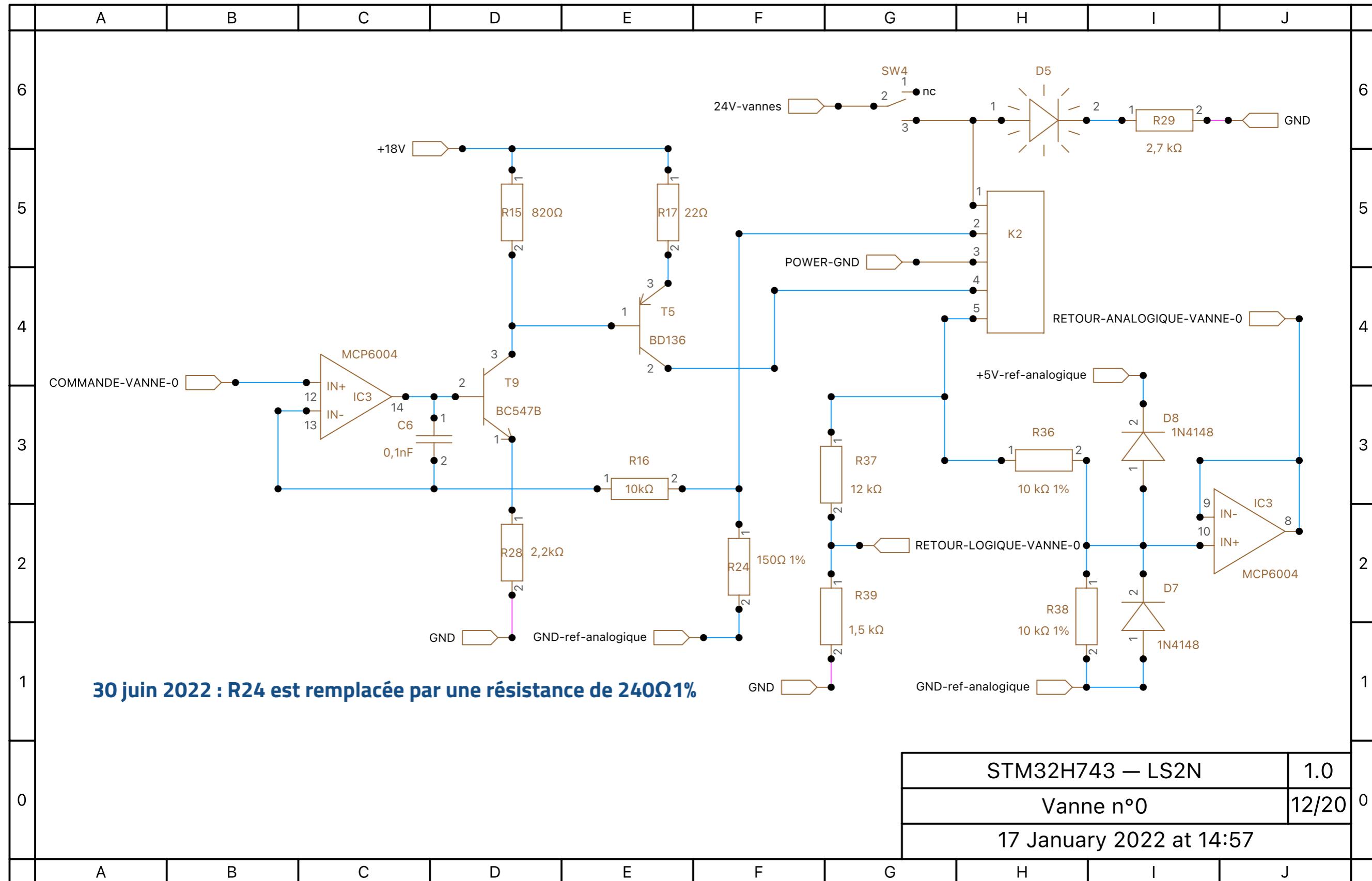
Plan de la carte (10/20)



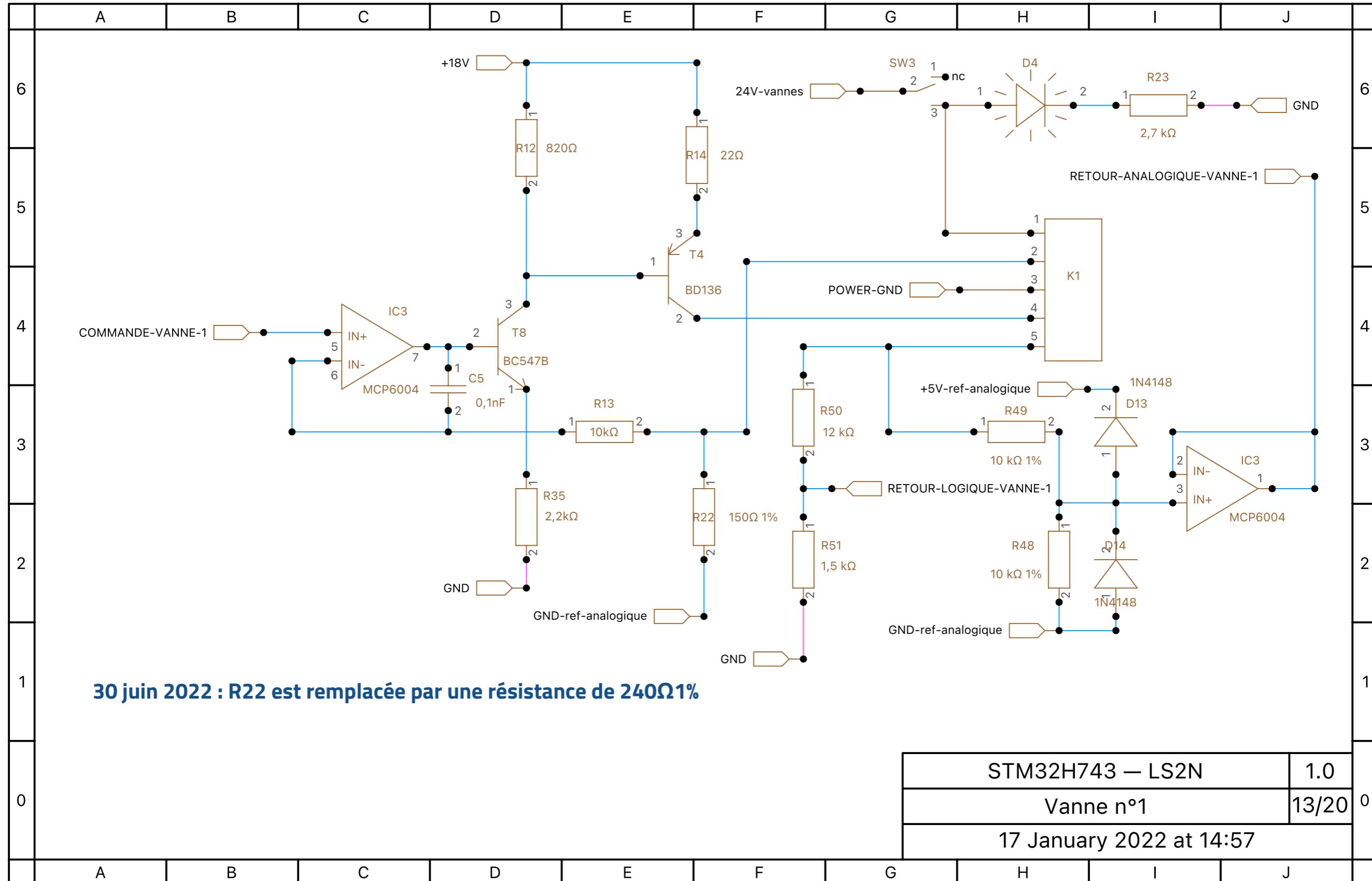
Plan de la carte (11/20)



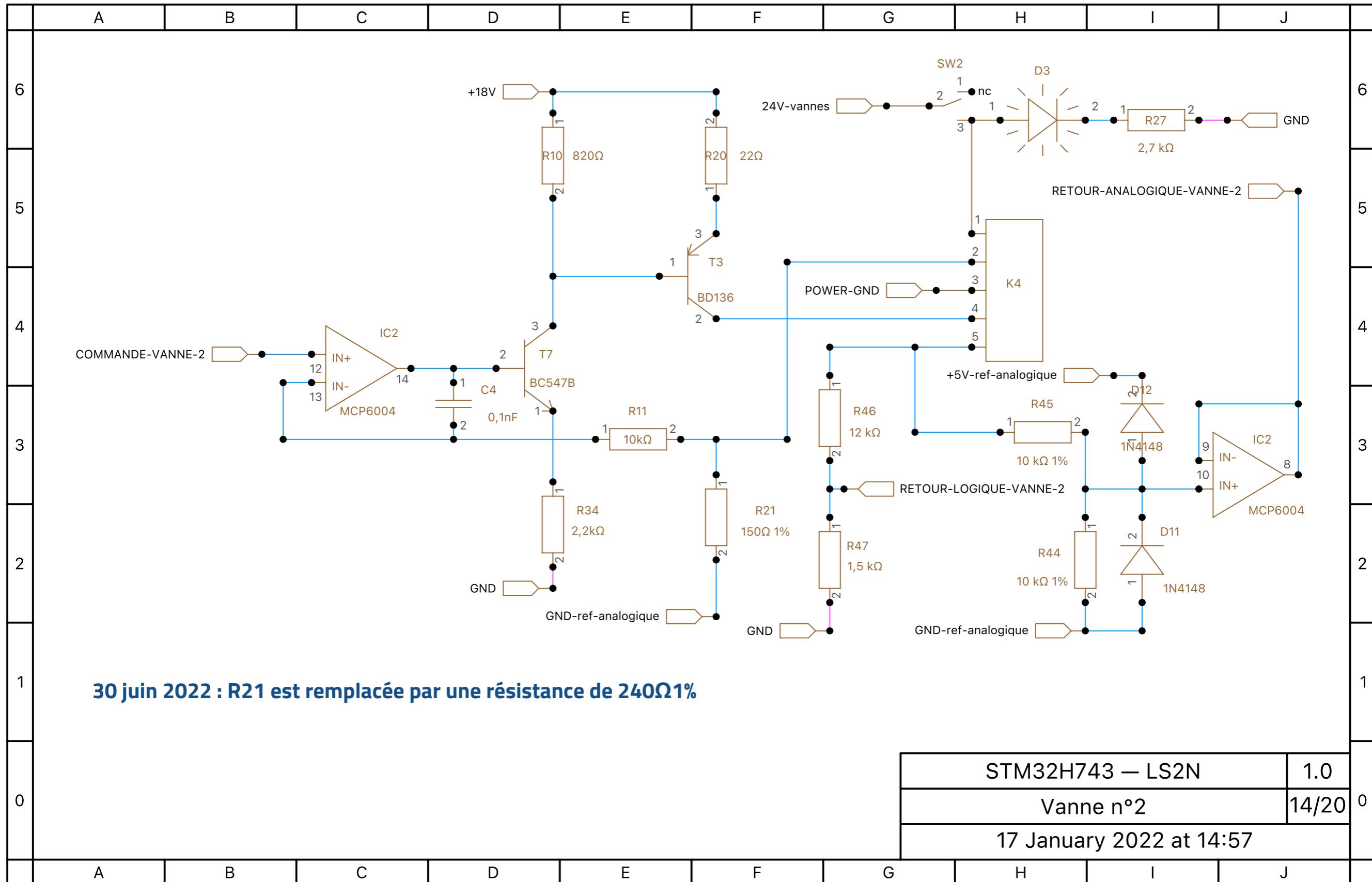
Plan de la carte (12/20)



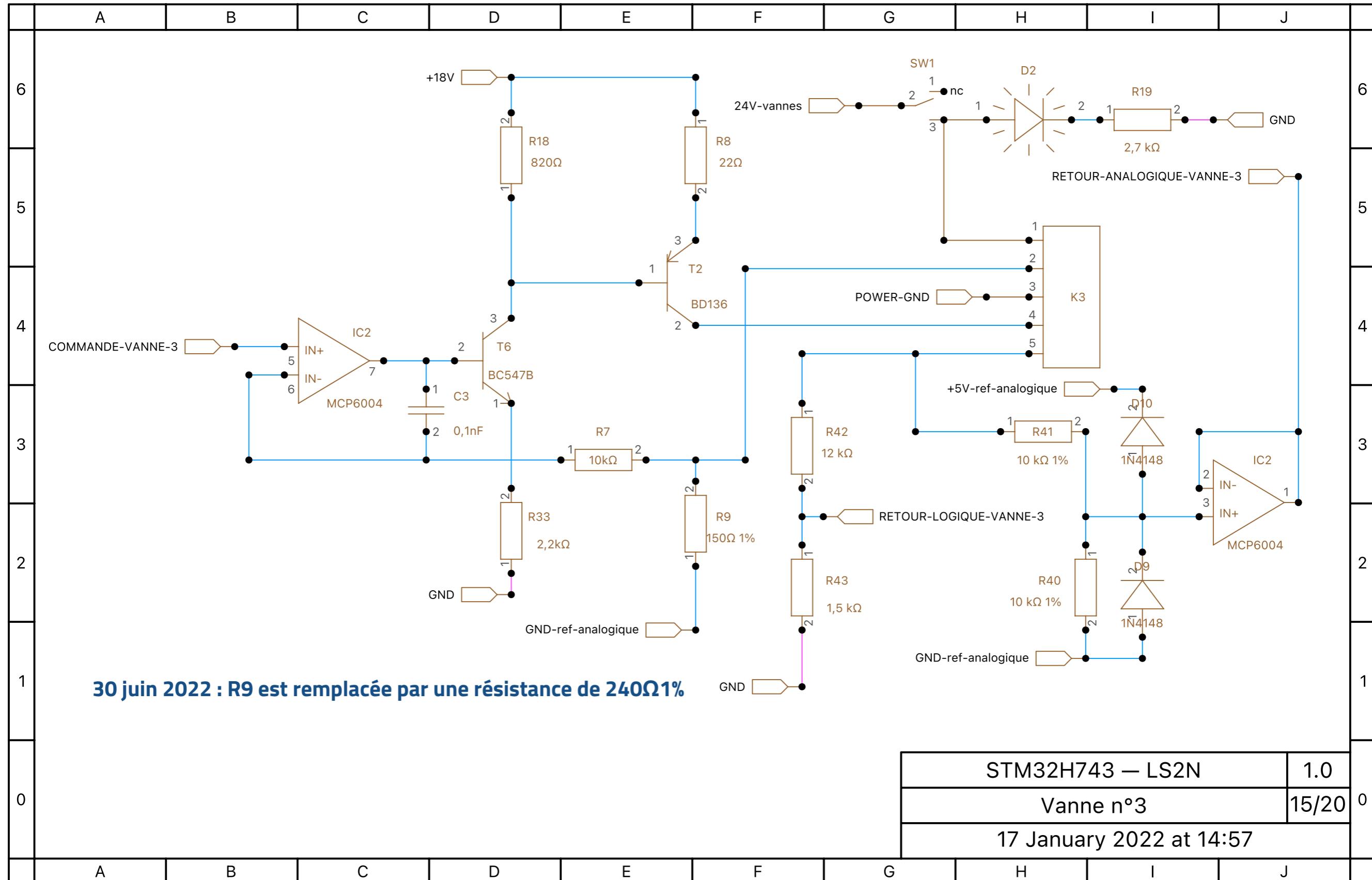
Plan de la carte (13/20)



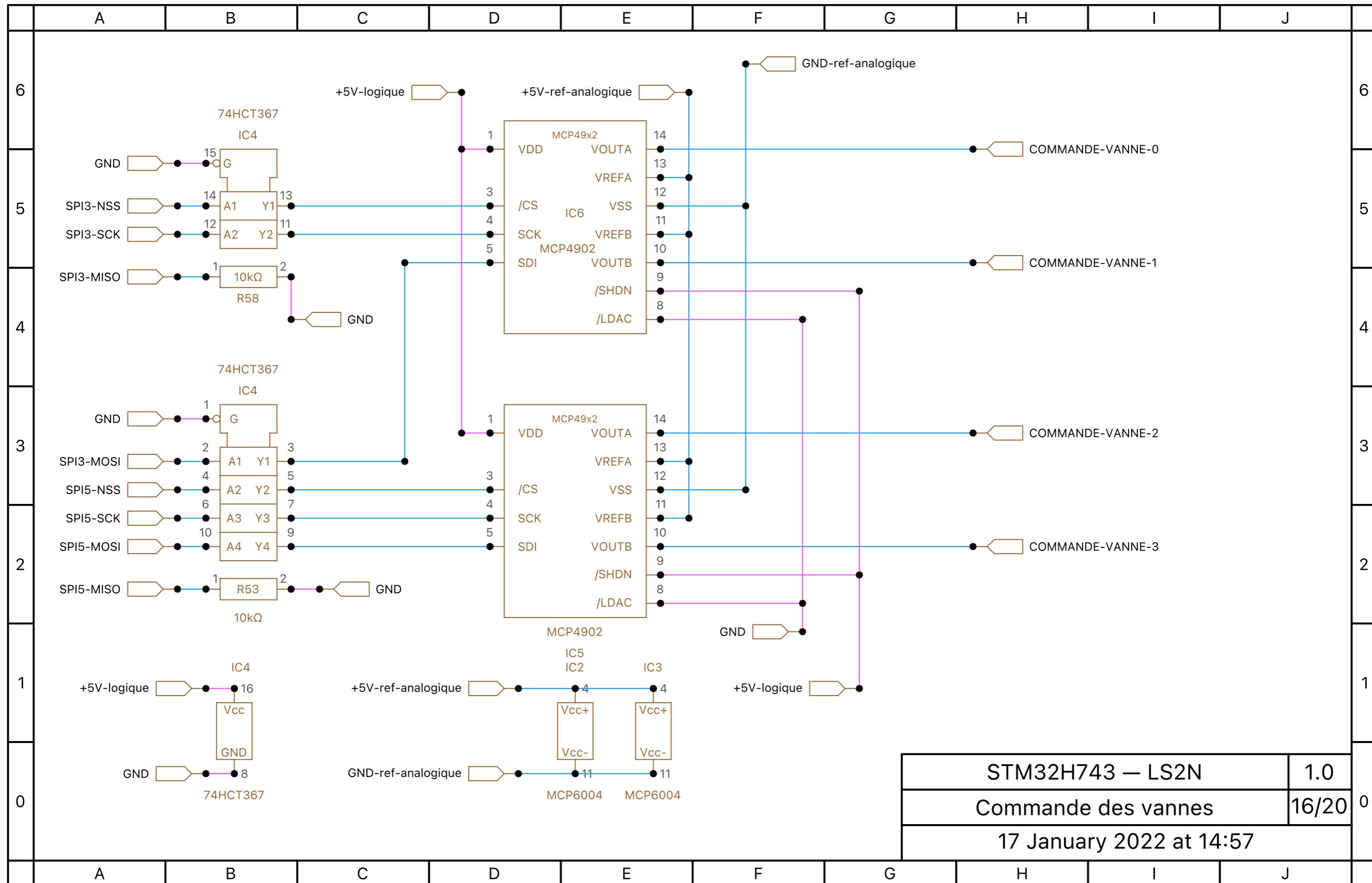
Plan de la carte (14/20)



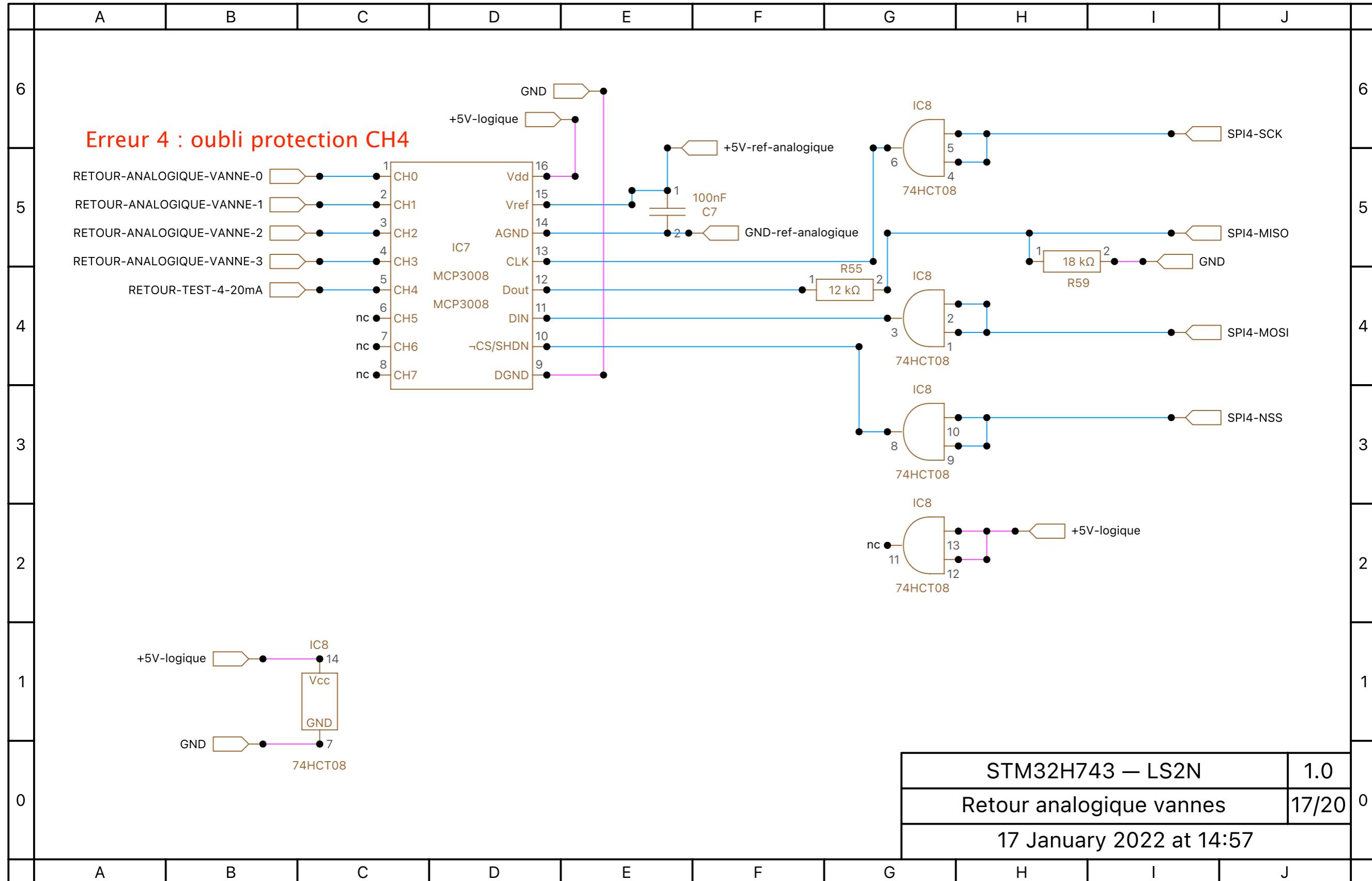
Plan de la carte (15/20)



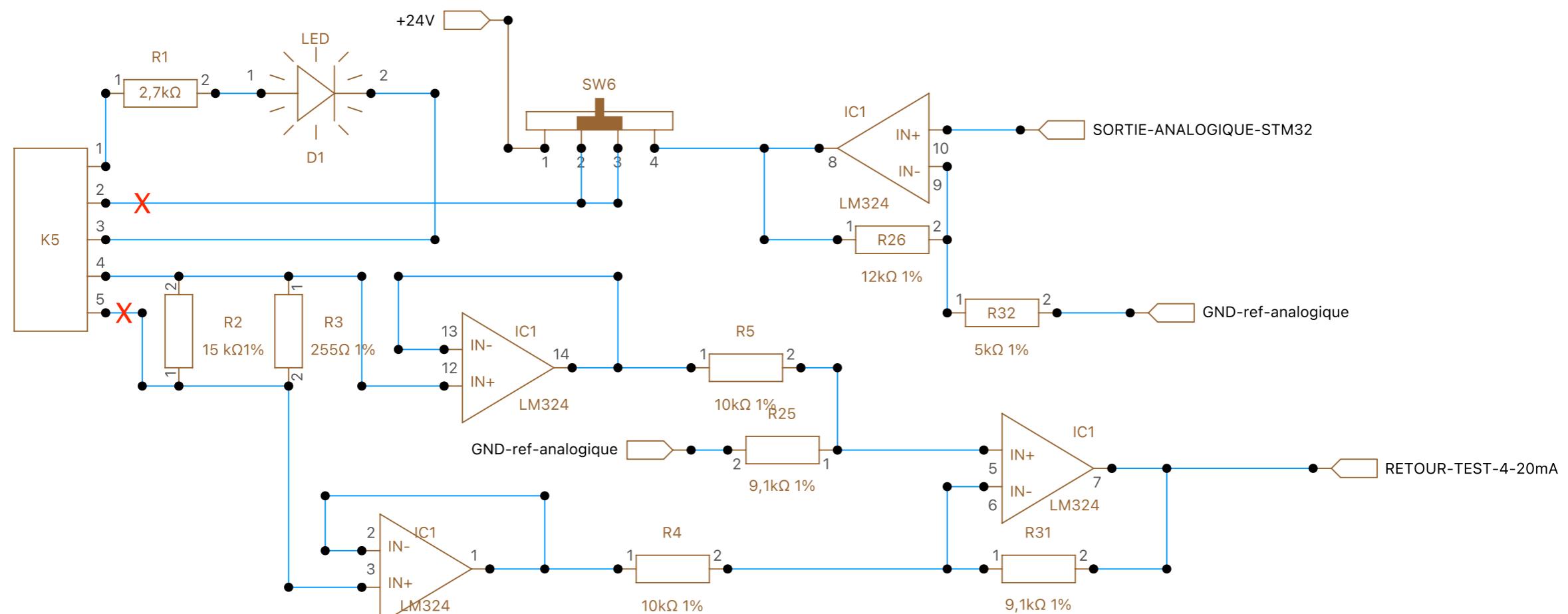
Plan de la carte (16/20)



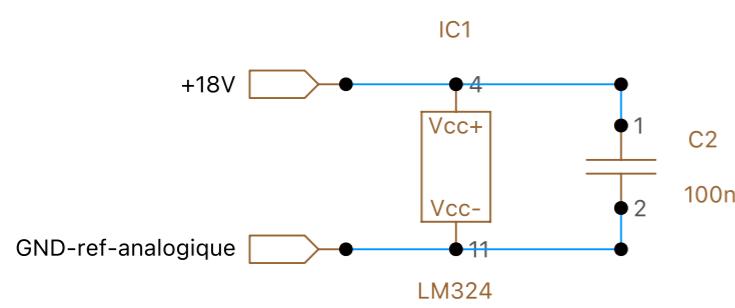
Plan de la carte (17/20)



Plan de la carte (18/20)



Erreurs 6 : ajouter une résistance de $1M\Omega$ entre les pattes 11 et 3 de IC1

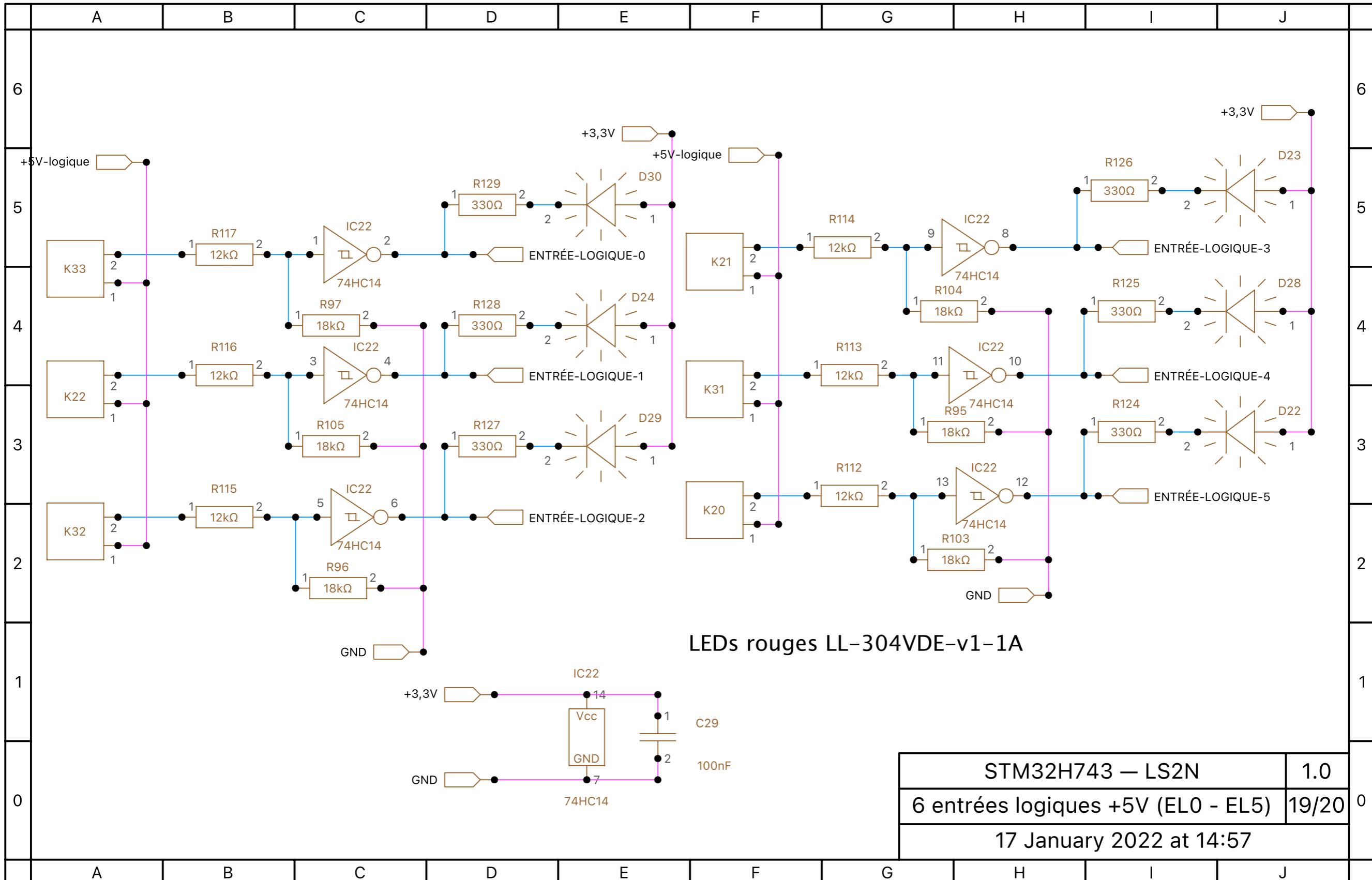


STM32H743 – LS2N

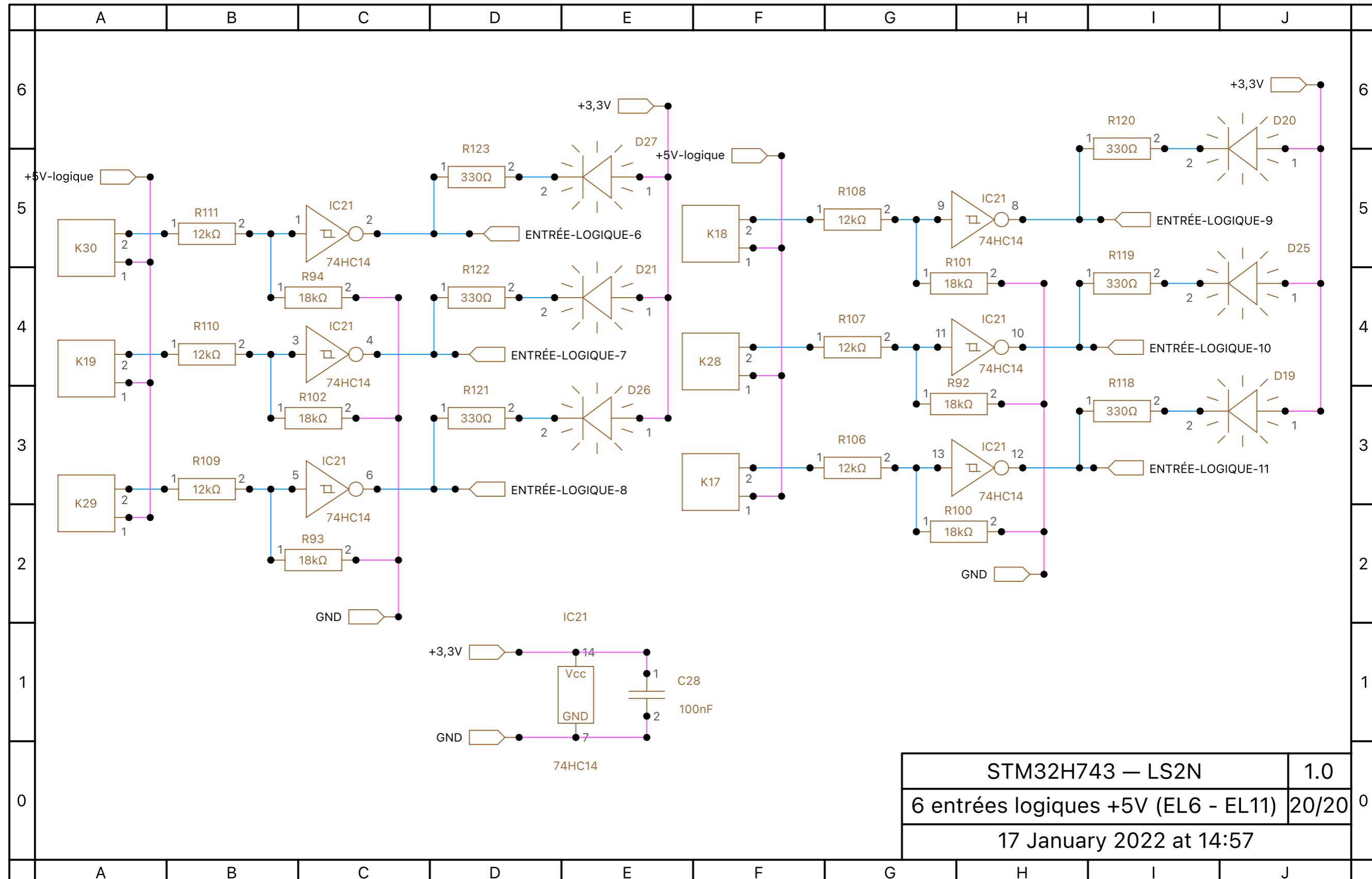
Test commande vannes

17 January 2022 at 14:57

Plan de la carte (19/20)



Plan de la carte (20/20)



STM32H743 – LS2N	1.0
6 entrées logiques +5V (EL6 - EL11)	20/20
17 January 2022 at 14:57	

Carte, vue de dessus

