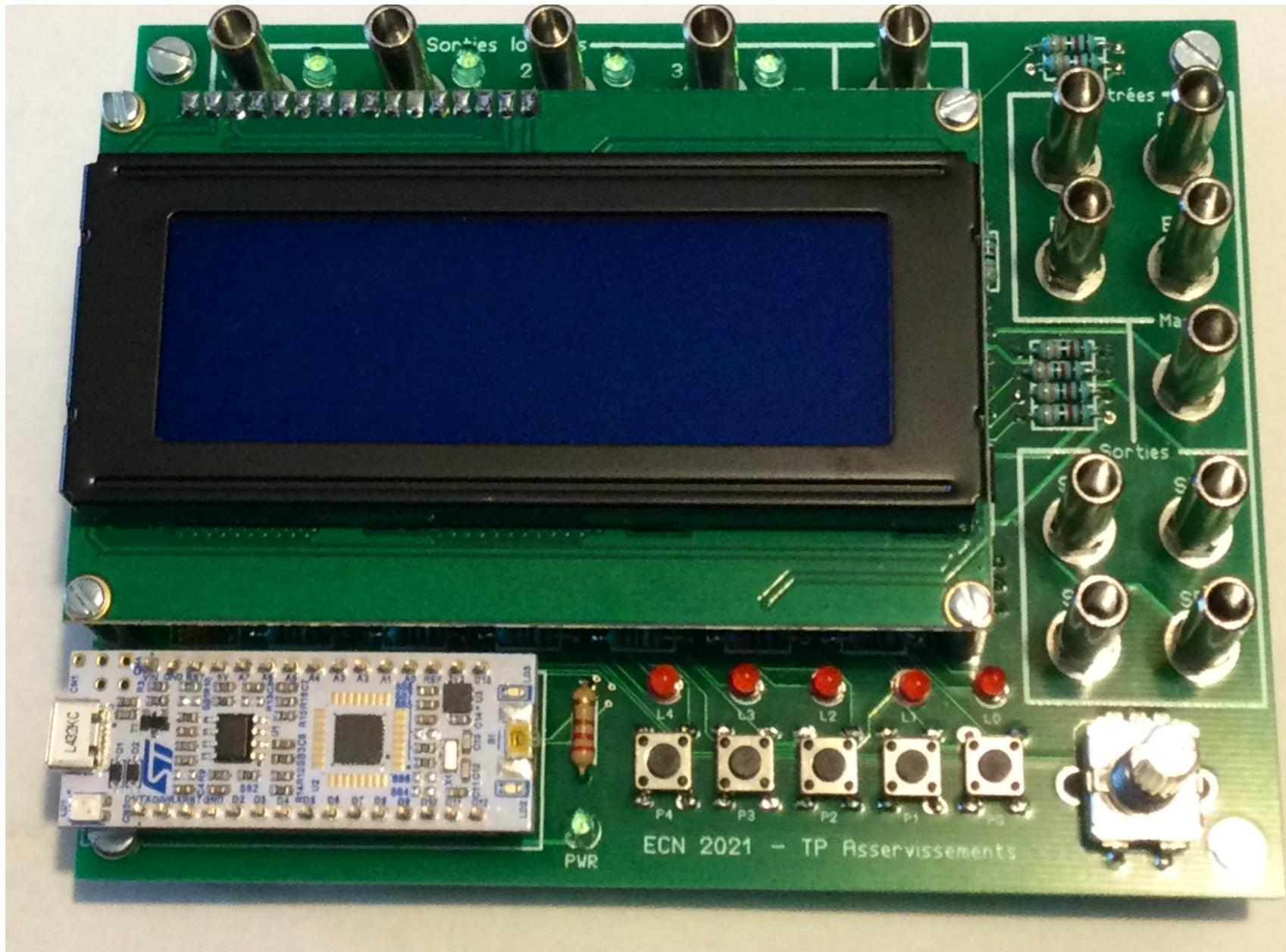


Carte Nucleo 32 asservissement



Pierre Molinaro

24 août 2021

Carte prototype Nucleo 32 asservissement

Le fichier *EICanari* de la carte, les croquis d'exemples, le fichier *keynote*, sont disponibles à l'URL :

<https://github.com/pierremolinaro/cartes-micro-controleurs-centrale-nantes/tree/master/carte-nucleo32-asservissement>

① Carte

Modules NUCLEO32 acceptés

La carte est conçue de façon à pouvoir être utilisée avec différents modules NUCLEO32. Les modules qui ont été testés avec succès sont :

- NUCLEO-F303K8 ;
- NUCLEO-G431KB ;
- NUCLEO-L432KC.

La carte accepte sans doute d'autres modules NUCLEO32.

Attention, il y a deux straps à enlever sur les modules (voir pages suivantes).

Module NUCLEO-G431KB

Attention, il y a deux straps à enlever sur le module NUCLEO-G431KB.

Il s'agit de SB2 (qui relie PA5 et PB7) et SB3 (qui relie PA6 et PA15), qui sont situés sur la face supérieure du module.

UM2397, page 17

SB3	ON	STM32 PA15 is connected to CN3 pin 7 for I2C SCL support on ARDUINO® Nano A5. In such a case, STM32 PA15 does not support ARDUINO® Nano D5 and PA6 must be configured as floating input.
	OFF	CN3 pin 7 is used as ARDUINO® Nano analog input A5 without I2C support and CN4 pin 8 is available as ARDUINO® Nano D5.
SB2	ON	STM32 PB7 is connected to CN3 pin 8 for I2C SDA support on ARDUINO® Nano A4. In such a case, STM32 PB7 does not support ARDUINO® Nano D4 and PA5 must be configured as floating input.
	OFF	CN3 pin 8 is used as ARDUINO® Nano analog input A4 without I2C support and CN4 pin 7 is available as ARDUINO® Nano D4.



Modules NUCLEO-F303K8 et NUCLEO-L432KC

Attention, il y a deux straps à enlever sur le modules NUCLEO-F303K8 et NUCLEO-L432KC. Il s'agit de SB16 (qui relie PA6 et PB6) et SB18 (qui relie PA5 et PB7), qui sont situés sous la carte.

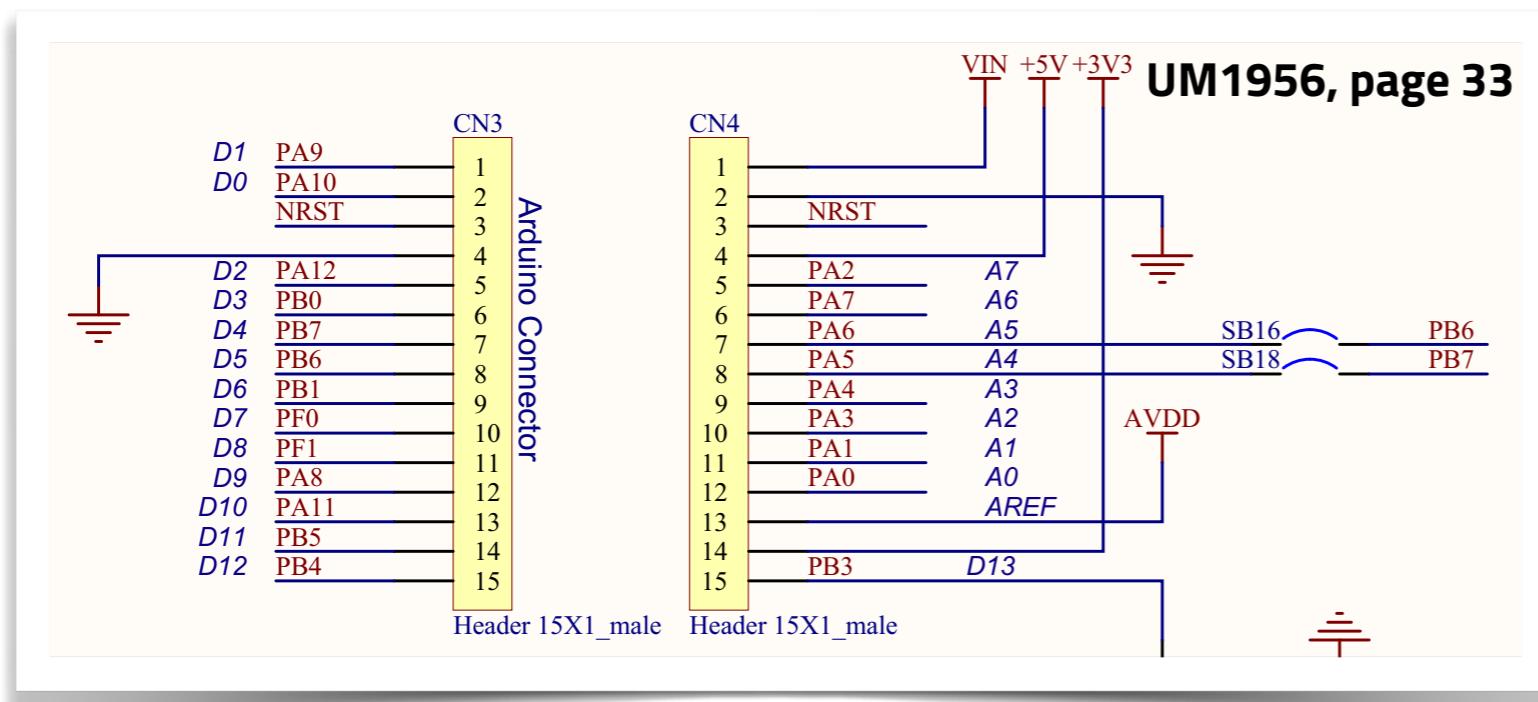


Figure 4. STM32 Nucl

UM1956, page 12

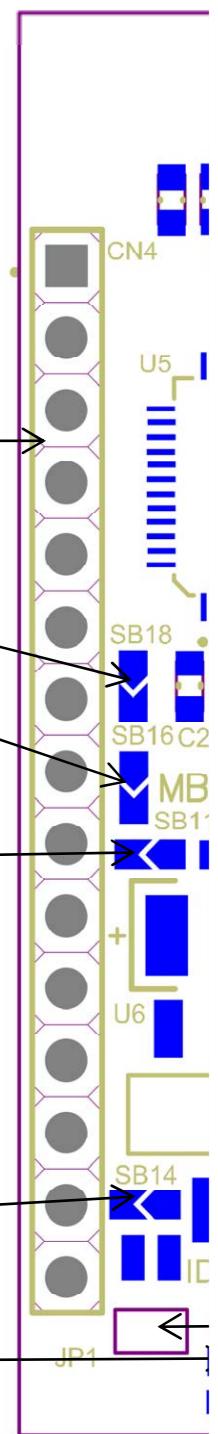
CN4
Arduino Nano connector

SB18
Connect D4 to A4
SB16
Connect D5 to A5

SB11
Connect STM32 pin 16 to
GND

SB14
3.3V regulator output

SB15
Connect D13 to LD3



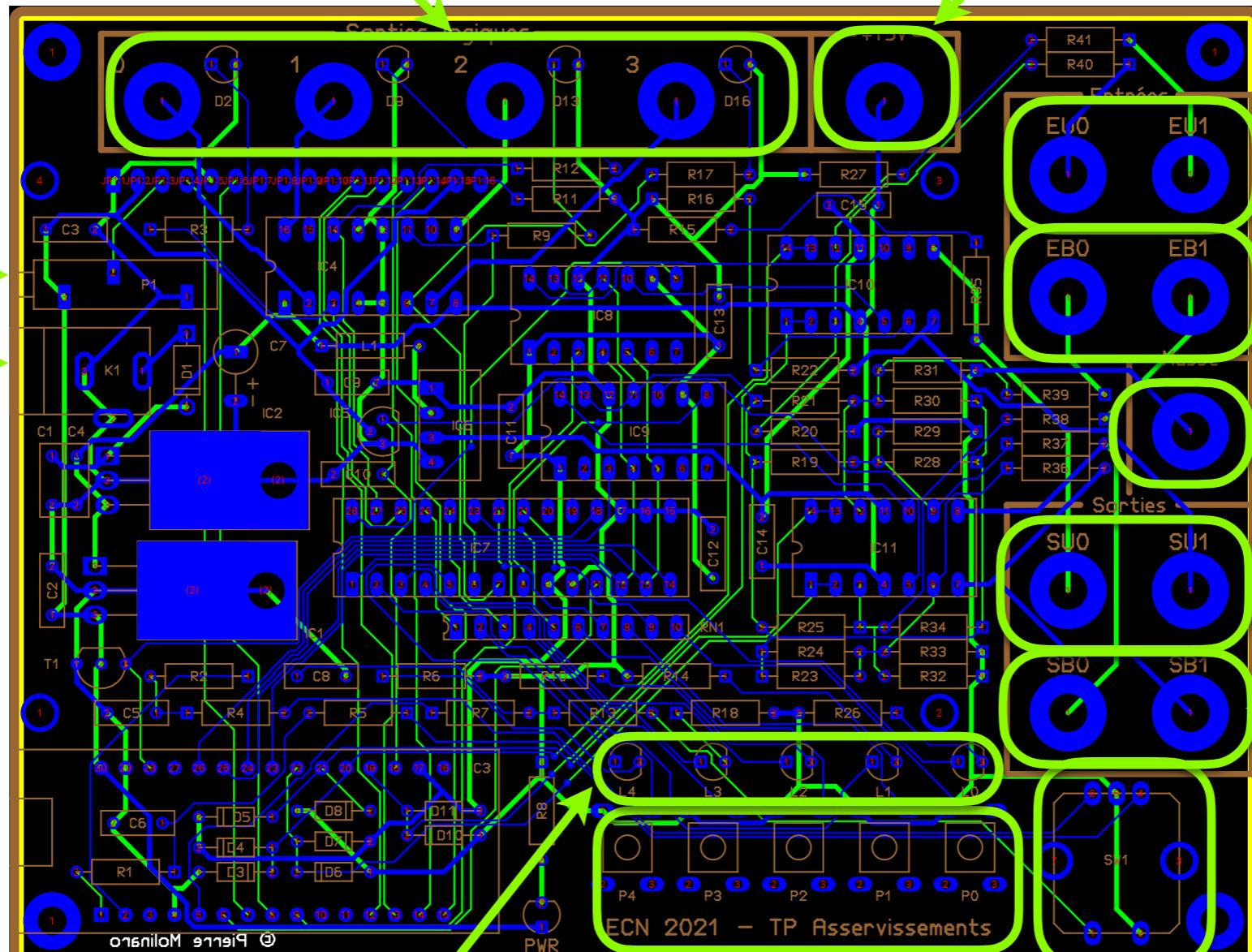
Circuit imprimé

4 sorties logiques de type *collecteur ouvert*, chacune accompagnée d'une led indiquant son activation.

Tension +15V,
fournie par la carte

Contraste LCD

Alimentation 18V



5 leds (L0 à L4)

5 poussoirs (P0 à P4)

Encodeur, avec poussoir CLIC

Cette led s'allume dès que la carte est alimenté (18V, et module Nucleo 32 présent).

Les fonctions d'entrées / sorties

Toutes les fonctions suivantes font l'objet des croquis d'exemple décrits dans la suite.

Afficheur LCD. C'est un afficheur de 4 lignes et 20 colonnes. Il est pris en charge par la librairie LiquidCrystal.

Leds. Les 5 leds L0 à L4 sont commandées par la fonction actionLed.

Poussoirs. On récupère l'état courant d'un des poussoirs P0 à P4 en appelant la fonction etatPoussoir.

Encodeur numérique. On récupère l'état courant du poussoir de l'encodeur en appelant la fonction etatClic. La plage des valeurs entières de l'encodeur est spécifiée par la fonction fixerGammeEncodeur, et on récupère la valeur courante de l'encodeur en appelant la fonction valeurEncodeur.

Sorties Logiques. Les quatre sorties logiques sont commandées individuellement, en appelant la fonction fixerSortieLogique.

Sorties analogiques unipolaires. Deux sorties analogiques fonctionnent en mode unipolaire (gamme [0V, 10V]) ; elles sont commandées en appelant actionSortieAnalogiqueUnipolaire.

Sorties analogiques bipolaires. Deux sorties analogiques fonctionnent en mode bipolaire (gamme [-10V, 10V]) ; elles sont commandées en appelant actionSortieAnalogiqueBipolaire.

Entrées analogiques unipolaires. Deux entrées analogiques fonctionnent en mode unipolaire (gamme [0V, 10V]) ; elles sont lues en appelant lireEntreeAnalogiqueUnipolaire.

Entrées analogiques. Deux entrées analogiques fonctionnent en mode bipolaire (gamme [-10V, 10V]) ; elles sont lues en appelant lireEntreeAnalogiqueBipolaire.

Emplacement et nom des composants

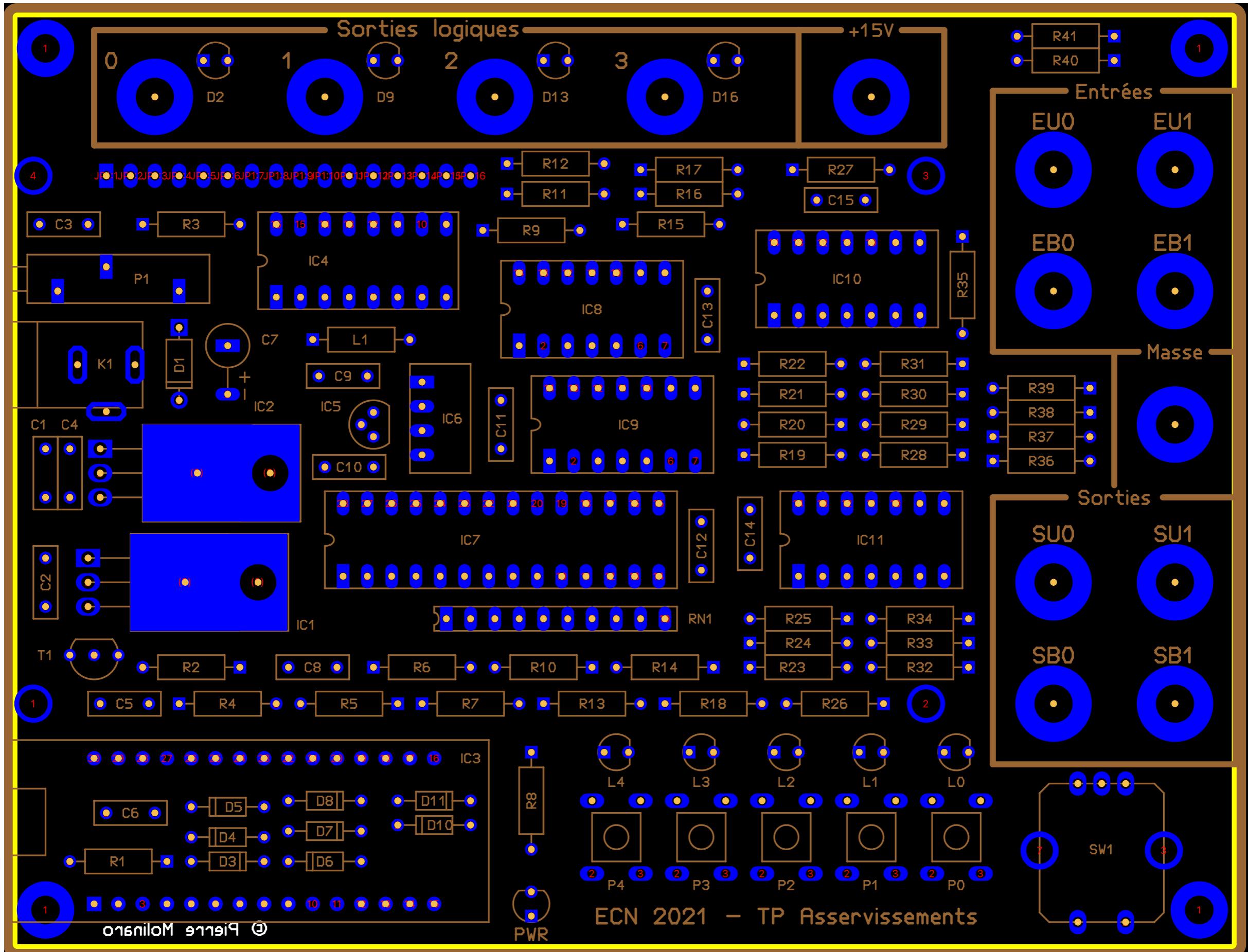


Schéma (1/9)

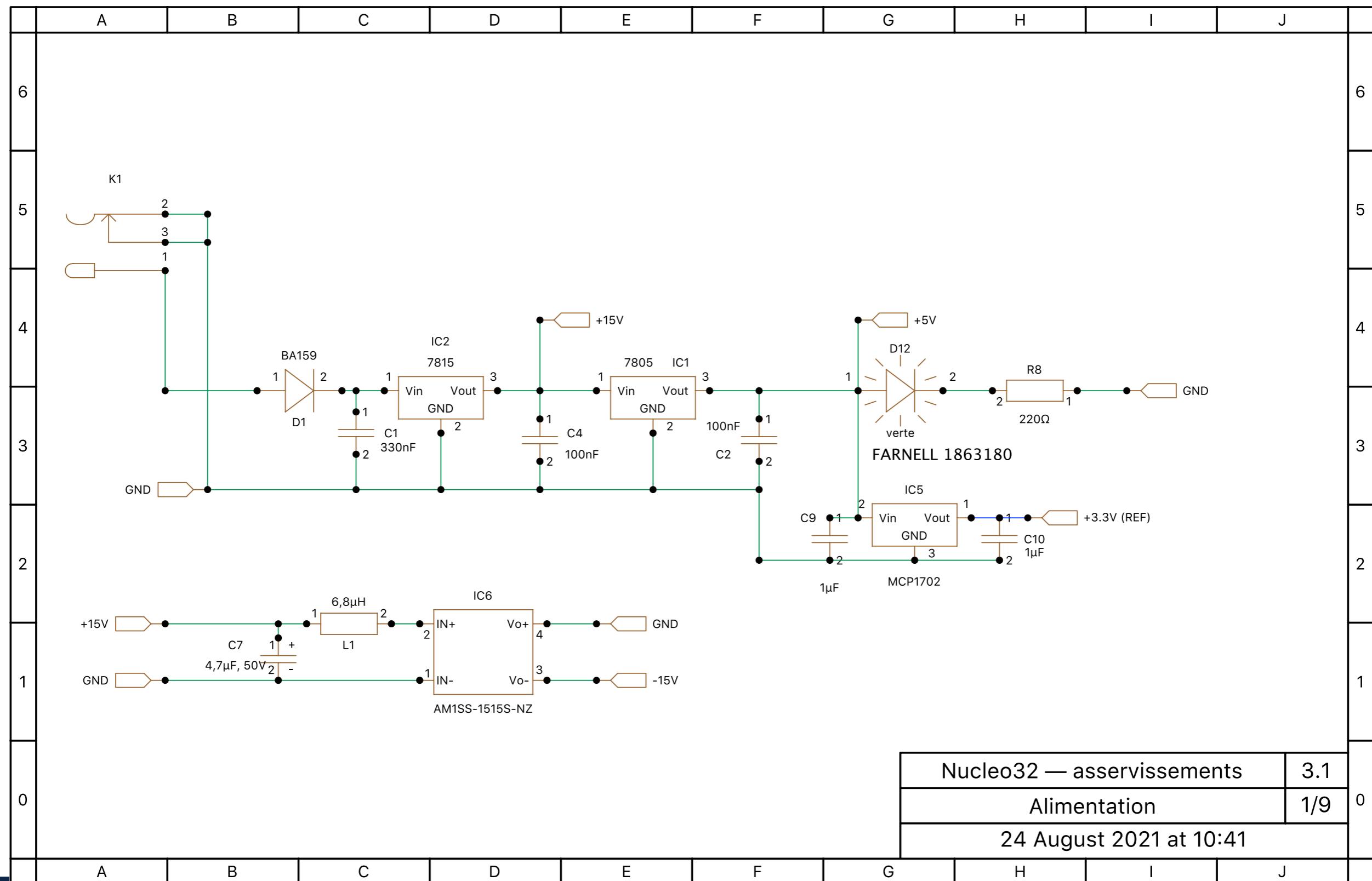


Schéma (2/9)

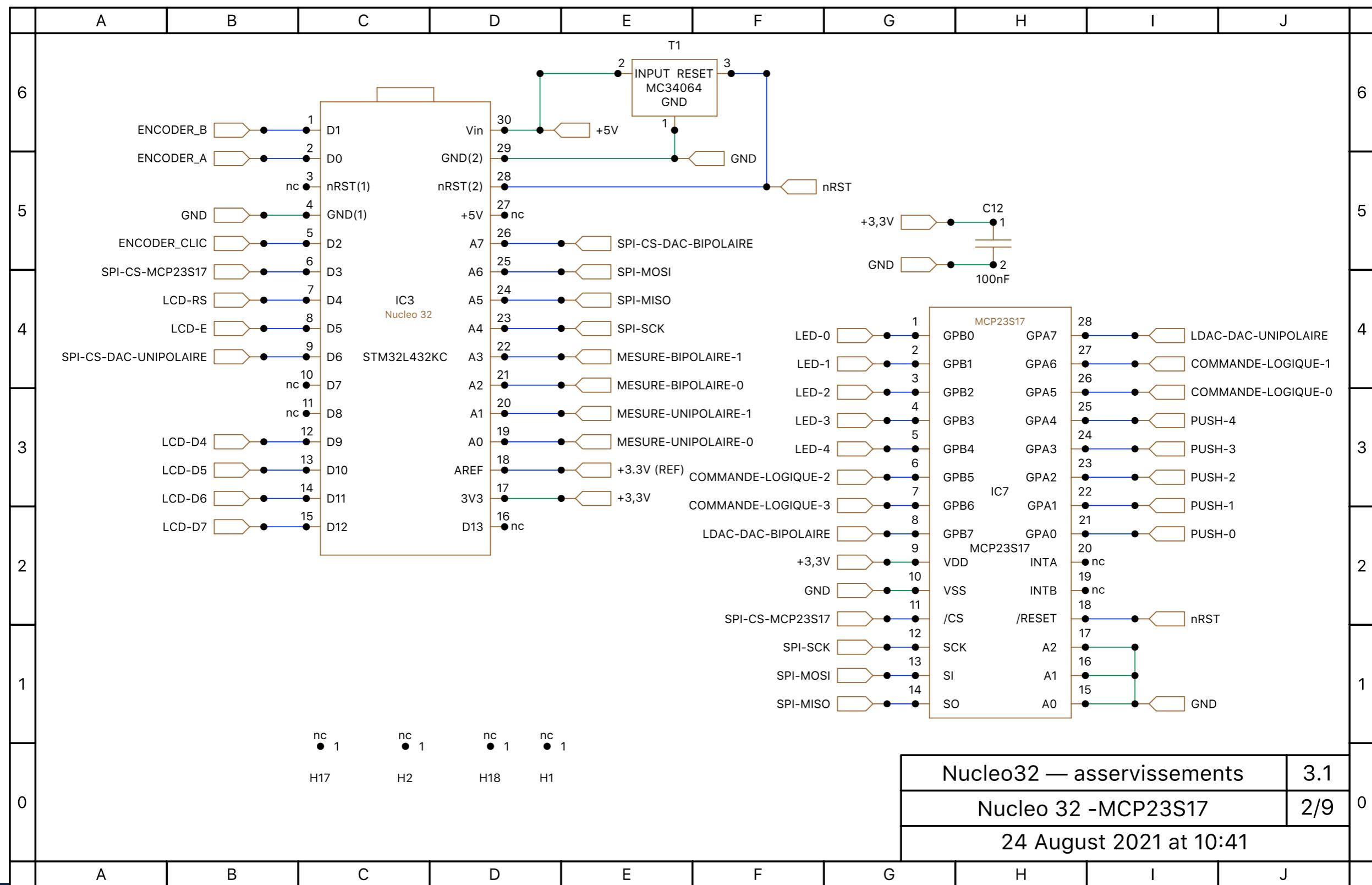


Schéma (3/9)

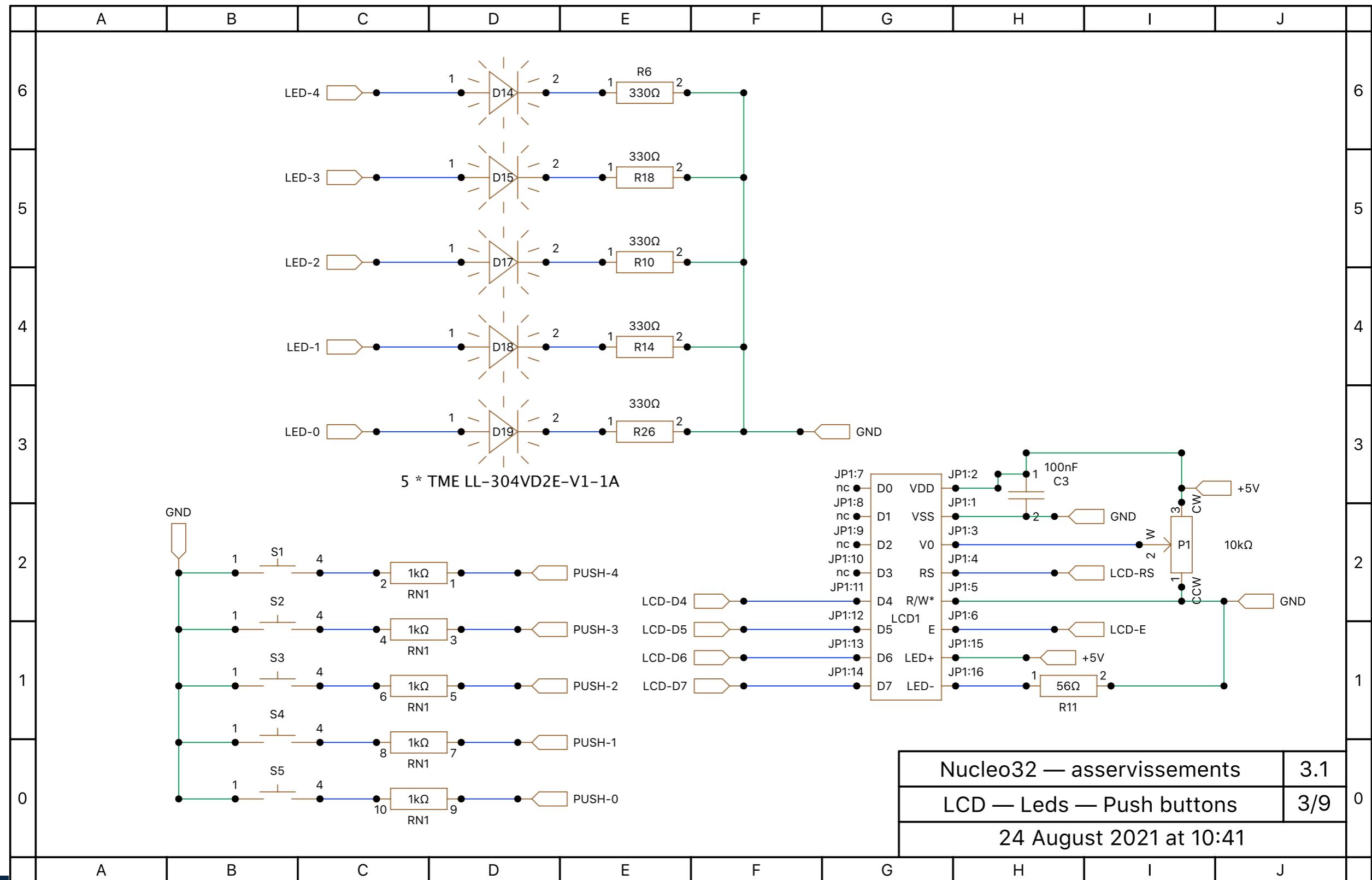


Schéma (4/9)

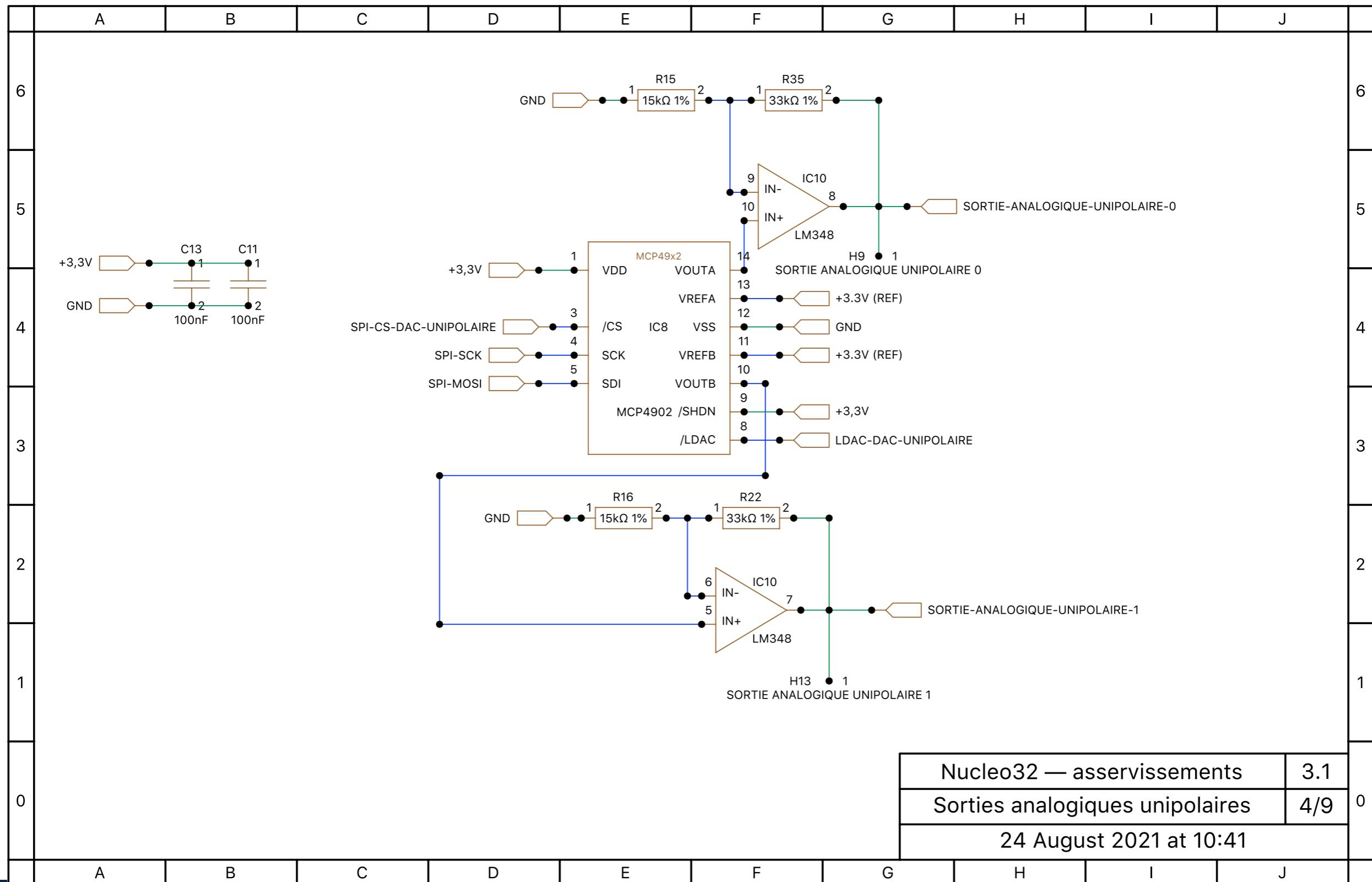


Schéma (5/9)

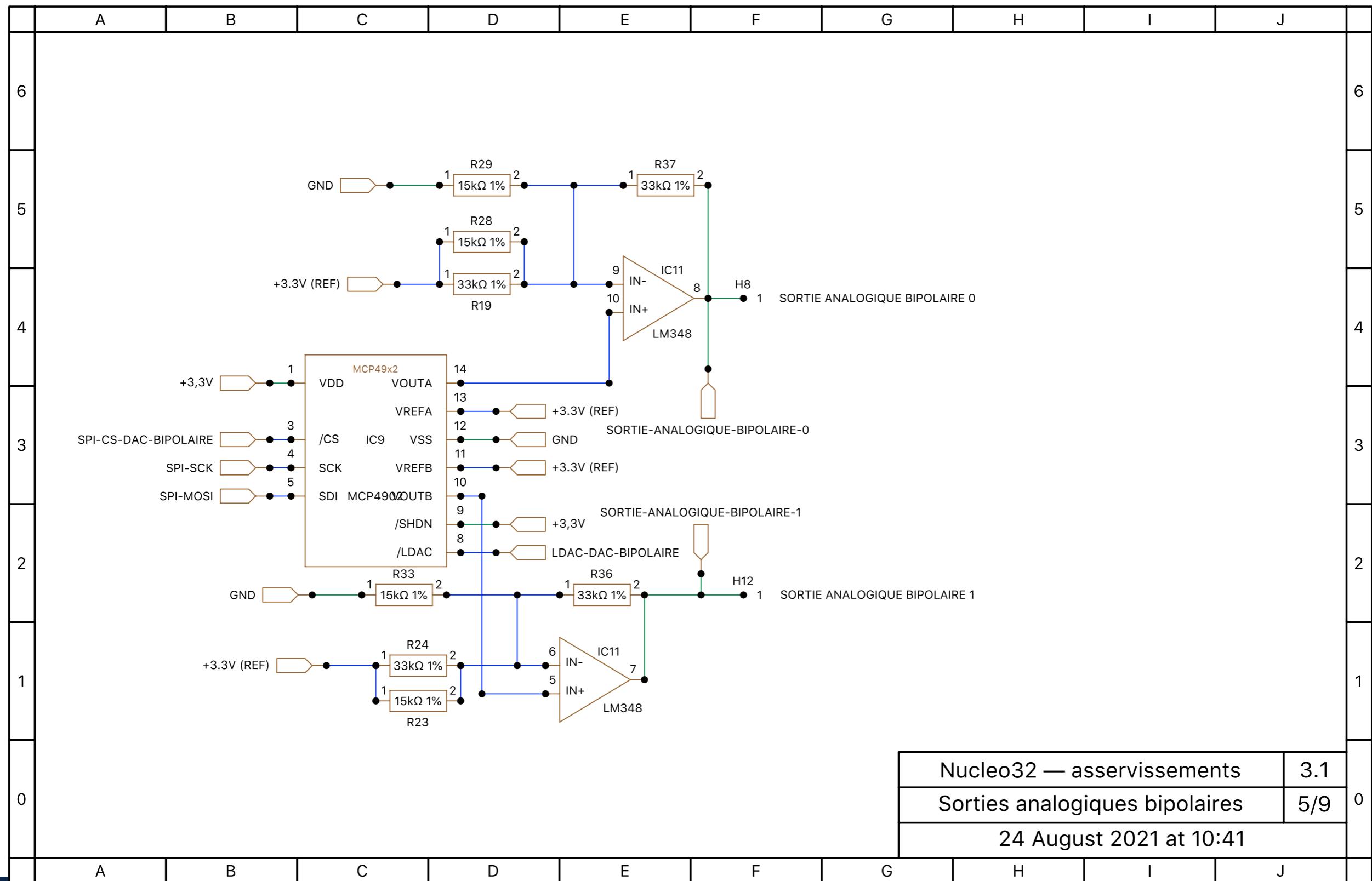


Schéma (6/9)

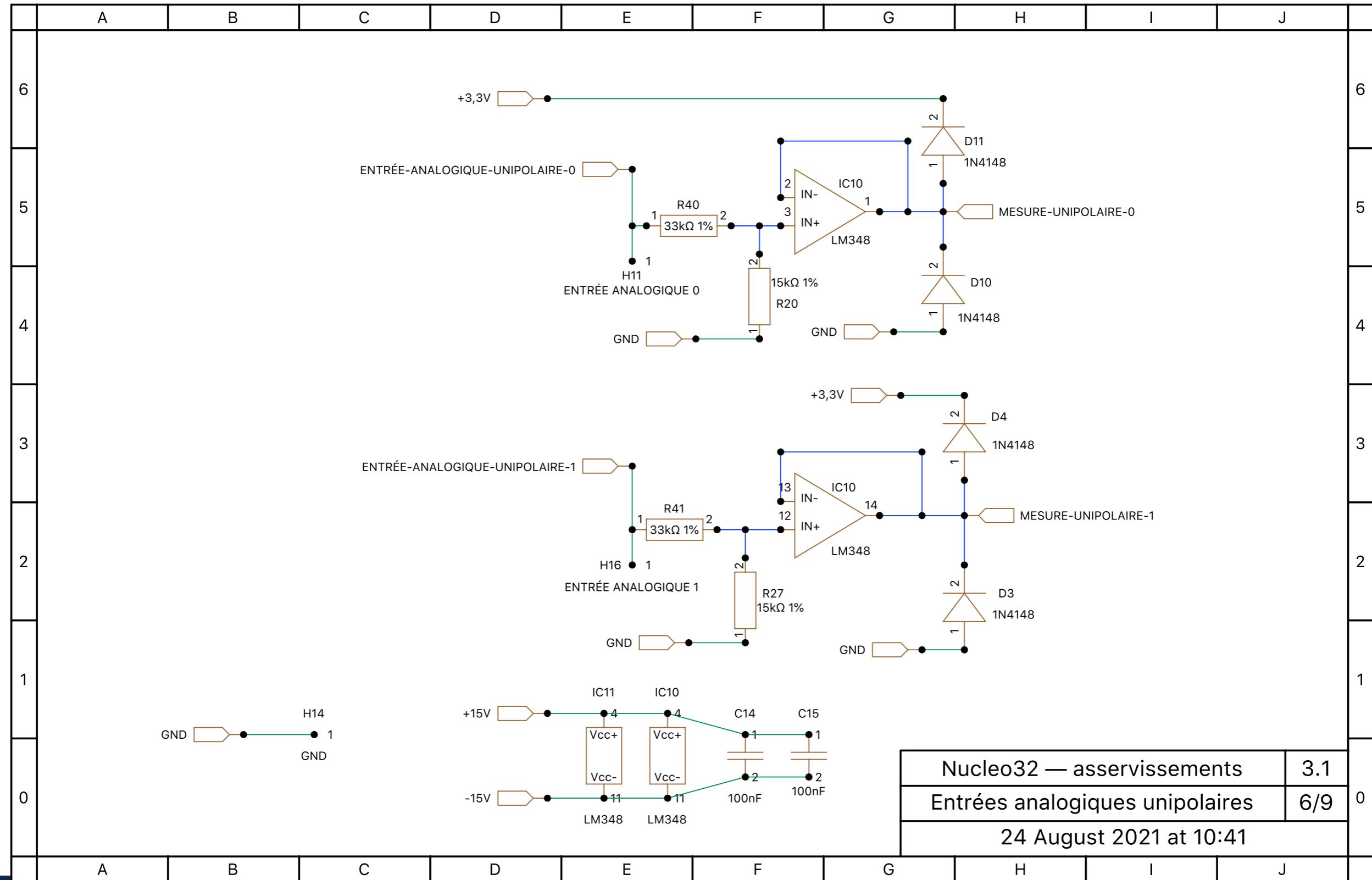


Schéma (7/9)

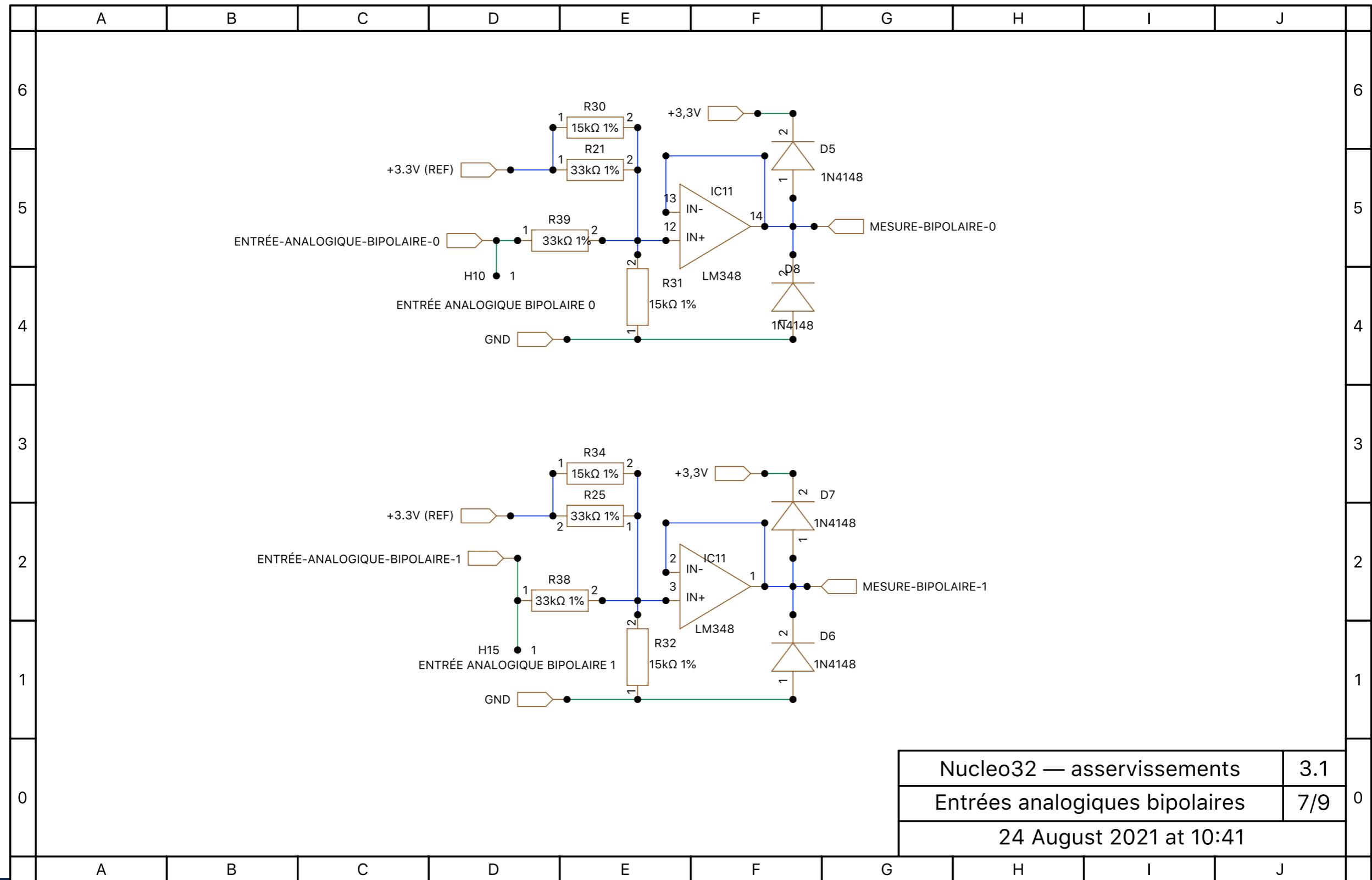


Schéma (8/9)

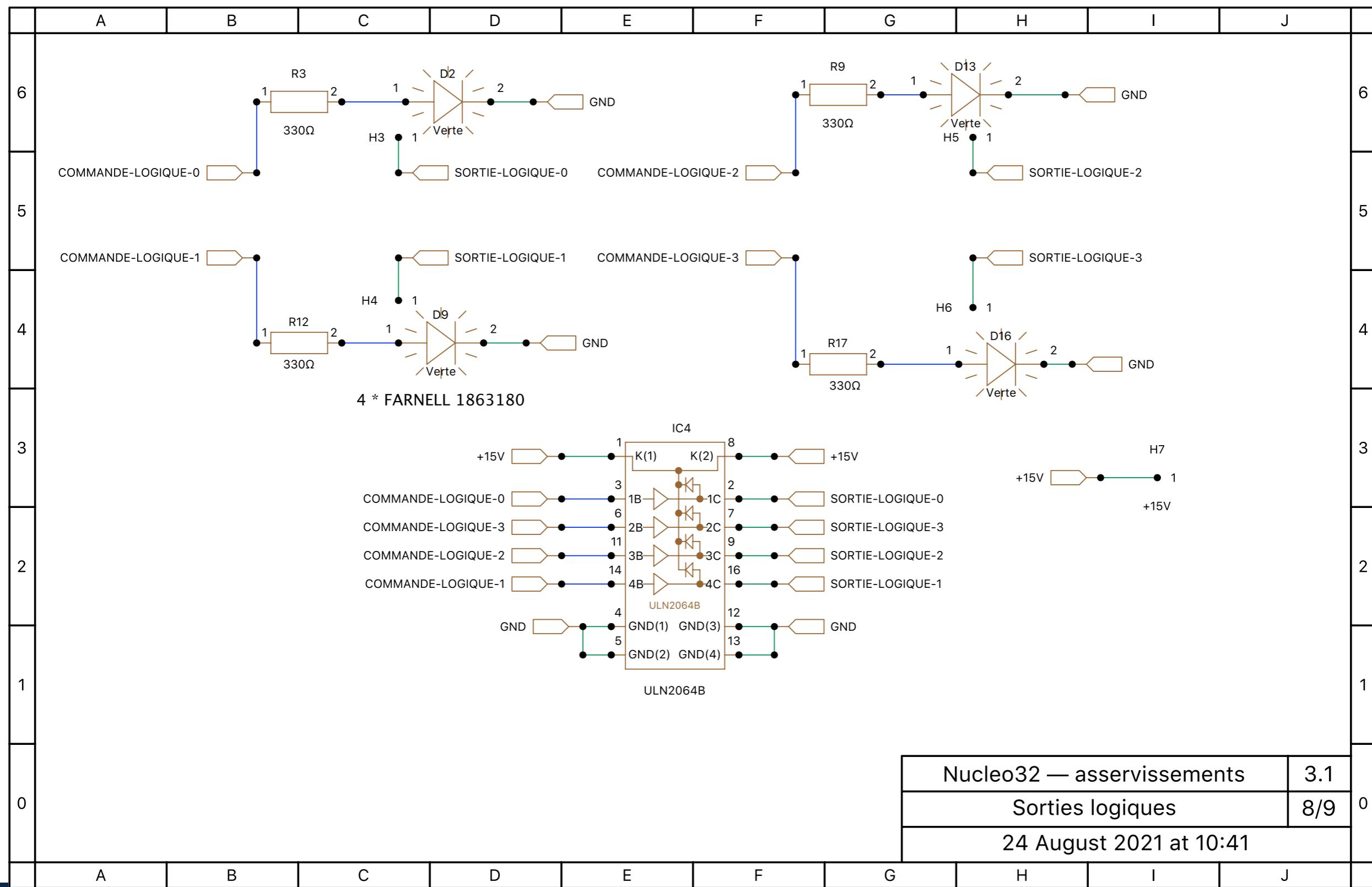
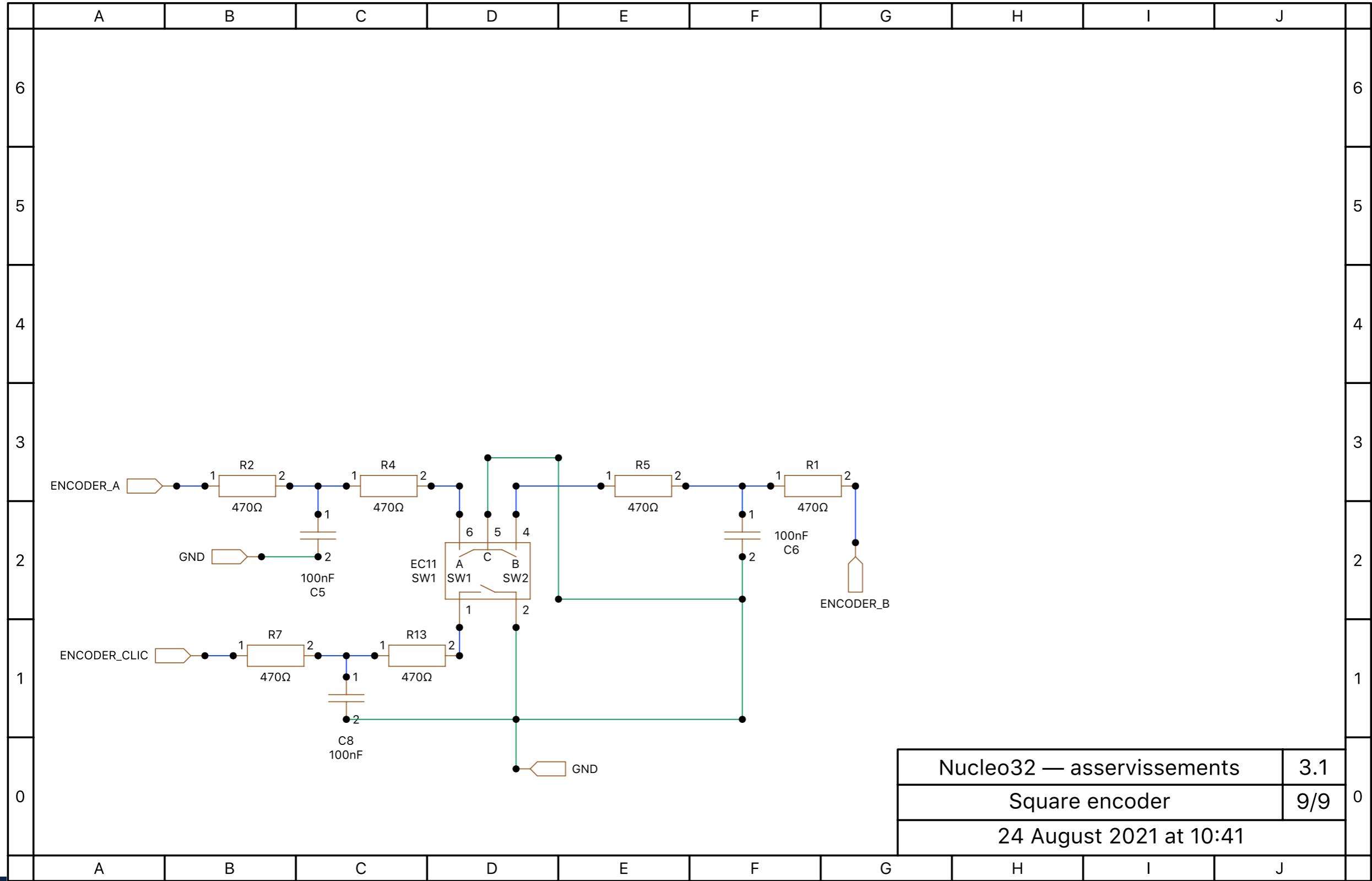


Schéma (9/9)



② STM32Duino

Configuration de l'environnement Arduino

Il faut installer le *gestionnaire de carte* dédié aux micro-contrôleurs STM32.

La page https://github.com/stm32duino/Arduino_Core_STM32, section *Getting Started* indique comment faire : ajouter le lien

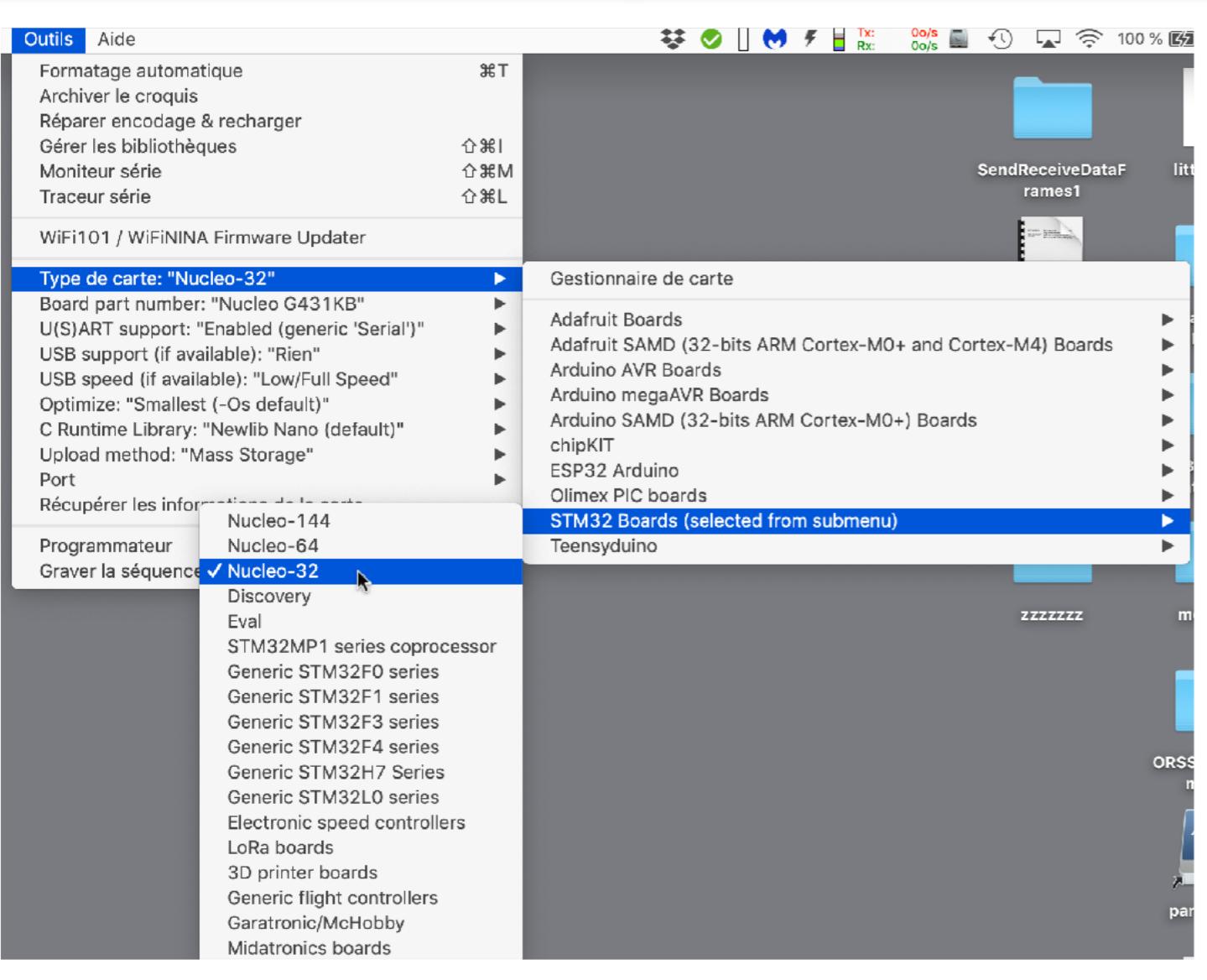
[https://github.com/stm32duino/BoardManagerFiles/raw/master/
package_stmicroelectronics_index.json](https://github.com/stm32duino/BoardManagerFiles/raw/master/package_stmicroelectronics_index.json)

dans le champ *URL de gestionnaire de cartes supplémentaires* de la page des préférences d'Arduino.

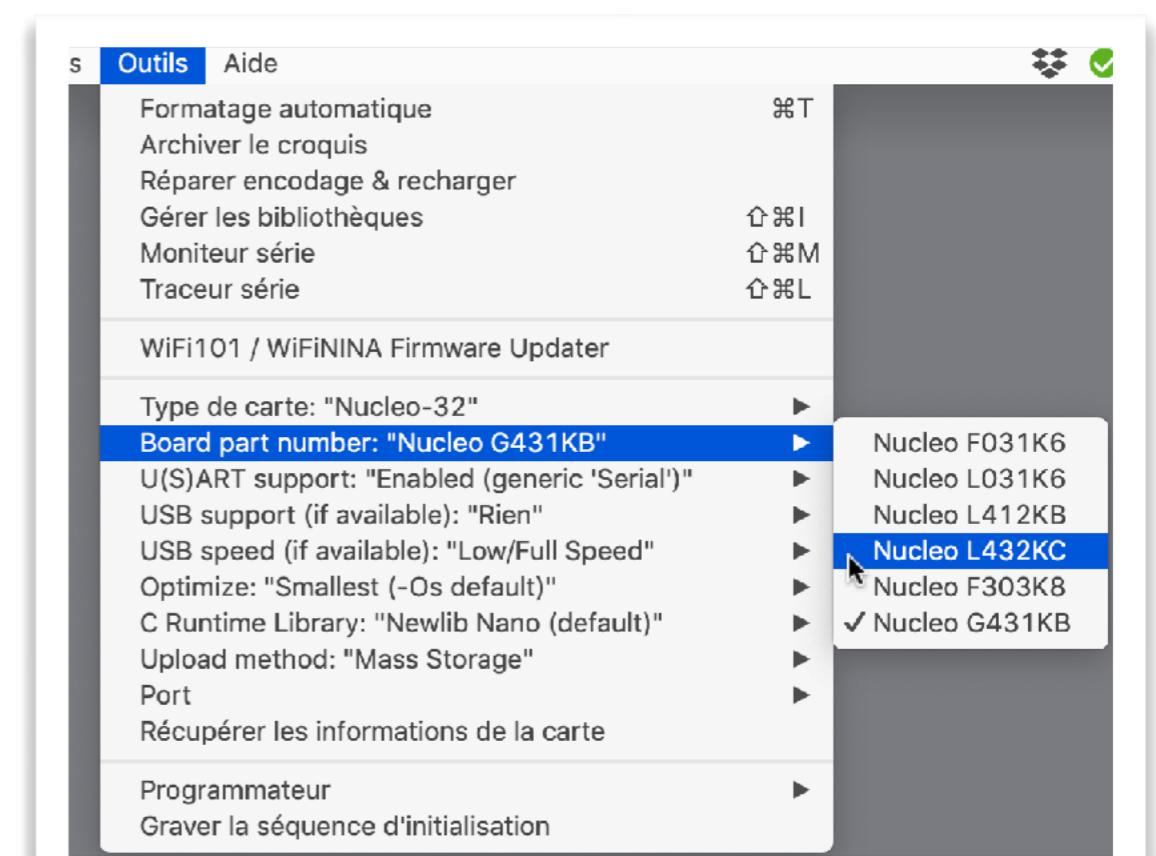
Sélection du module NUCLEO-L432KC dans Arduino

Après l'installation du *gestionnaire de carte* dédié aux micro-contrôleurs STM32, il faut sélectionner le module NUCLEO-L432KC. Avec Arduino version 1.8.15, ceci se fait en deux étapes.

① Sélectionner « Nucleo-32 »



② Sélectionner « Nucleo L432KC »



Ci dessus, la carte Nucleo G431KB était sélectionnée au départ. Quand on relâche la souris, la carte Nucleo L432KC devient la carte sélectionnée.

③ Croquis d'exemple

Constitution d'un croquis

Tout le code de gestion de la carte *asservissement* est contenu dans deux fichiers :

- le fichier d'en-tête `carte-asservissement.h` ;
- le fichier d'implémentation `carte-asservissement.cpp`.

Pour les utiliser dans un croquis, il y a 3 possibilités.

- ① Les inclure simplement dans le répertoire du croquis, à côté du fichier `.ino`.
- ② Inclure les *liens symboliques* vers ces fichiers (Mac, Linux). 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/carte-asservissement.cpp .
ln -s chemin/carte-asservissement.h .
```

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

- ③ Créer une librairie Arduino.

Écriture d'un croquis

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

- ① Inclure `carte-asservissement.h` au début du fichier :

```
#include "carte-asservissement.h"
```

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

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

- ③ Ensuite, toutes les fonctions citées dans `carte-asservissement.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.

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 : par exemple, faire clignoter une led à 1 Hz, un premier asservissement à 2 Hz, un deuxième à 3 Hz, ...

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

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.

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-leds-et-poussoirs (1/2)

Ce croquis montre comment commander les leds et lire les poussoirs.

Pour commander une led, appeler `actionLed` :

```
actionLed (nomLed, valeur) ;
```

où :

- *nomLed* est `LED::L0`, `LED::L1`, ..., `LED::L4` pour désigner une des cinq leds ;
- *valeur* est **false** si on veut éteindre la led, **true** si on veut l'allumer.

Pour lire l'état d'un poussoir, appeler `etatPoussoir` :

```
etatPoussoir (nomPoussoir) ;
```

où :

- *nomPoussoir* est `POUSSOIR::P0`, `POUSSOIR::P1`, ..., `POUSSOIR::P4` pour désigner un des cinq poussoirs.

La fonction `etatPoussoir` renvoie **false** si le poussoir est relâché, **true** si il est appuyé.

Croquis 03-leds-et-poussoirs (2/2)

Le croquis utilise `actionLed` et `etatPoussoir` de la façon suivante :

```
actionLed (LED::L0, etatPoussoir (POUSSOIR::P0)) ;  
actionLed (LED::L1, etatPoussoir (POUSSOIR::P1)) ;  
actionLed (LED::L2, etatPoussoir (POUSSOIR::P2)) ;  
actionLed (LED::L3, etatPoussoir (POUSSOIR::P3)) ;  
actionLed (LED::L4, etatPoussoir (POUSSOIR::P4)) ;
```

C'est-à-dire que chaque led reflète l'état du poussoir correspondant :

- le poussoir est relâché, la led est éteinte ;
- le poussoir est appuyé, la led est allumée.

Croquis 04-encodeur-et-clic (1/3)

Le croquis montre l'utilisation de l'encodeur numérique :

- prise en charge de sa rotation (20 crans par tour) ;
- prise en charge de son poussoir.

Le plus simple : la fonction sans argument `etatClic` renvoie :

- la valeur `false` si le poussoir de l'encodeur est relâché ;
- la valeur `true` si le poussoir de l'encodeur est appuyé.

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.

Croquis 04-encodeur-et-clic (2/3)

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-encodeur-et-clic (3/3)

Le croquis surveille le clic de l'encodeur, et affiche son état (appuye / relache) sur la deuxième ligne de l'afficheur.

La gamme de l'encodeur est fixée à [-10, 10], par un appel à `fixerGammeEncodeur` dans la fonction `setup`. La fonction `loop` affiche la valeur de l'encodeur sur la troisième ligne.

Croquis 05-sorties-logiques (1/2)

Ce croquis montre comment commander les sorties logiques.

Les quatre sorties logiques sont de type collecteur ouvert, c'est-à-dire que chacune d'elles est dans l'un de deux états suivants :

- *inactive* ou *flottant*, aucun courant ne circule dans la sortie (la led correspondante est éteinte) ;
- *active*, la tension de sortie est proche de zéro, le courant est entrant (la led correspondante est allumée).

Pour commander une sortie logique, appeler `actionSortieLogique` :

```
actionSortieLogique (nomSortieLogique, valeur) ;
```

où :

- *nomSortieLogique* est `SORTIE_LOGIQUE::S0`, `SORTIE_LOGIQUE::S1`, ..., `SORTIE_LOGIQUE::S4` pour désigner une des quatre sorties logiques ;
- *valeur* est `false` si on veut désactiver la sortie (led éteinte), `true` si on veut l'activer (led allumée).

Croquis 05-sorties-logiques (2/2)

Le croquis utilise `actionSortieLogique` et `etatPoussoir` de la façon suivante :

```
actionSortieLogique (SORTIE_LOGIQUE::S0, etatPoussoir (POUSSOIR::P0)) ;  
actionSortieLogique (SORTIE_LOGIQUE::S1, etatPoussoir (POUSSOIR::P1)) ;  
actionSortieLogique (SORTIE_LOGIQUE::S2, etatPoussoir (POUSSOIR::P2)) ;  
actionSortieLogique (SORTIE_LOGIQUE::S3, etatPoussoir (POUSSOIR::P3)) ;
```

C'est-à-dire que chaque sortie logique reflète l'état du poussoir correspondant :

- le poussoir est relâché, la sortie logique correspondante est inactive (led éteinte) ;
- le poussoir est appuyé, la sortie logique correspondante est active (led allumée).

Croquis 06-es-analogiques-unipolaires (1/4)

Ce croquis montre comment commander les sorties analogiques unipolaires (c'est-à-dire la gamme [0, 10V]), et lire les entrées analogiques unipolaires (c'est-à-dire la gamme [0, 10V]).

Fonctionnement :

- l'encodeur fixe la tension des deux sorties analogiques (0 → 0V, 255 → 10,56V):
- la valeur des sorties est affichée ;
- les entrées analogiques sont mesurées et affichées.

Pour vérifier le bon fonctionnement, il est utile de se munir de fils pour relier une des deux sorties analogiques unipolaires aux entrées analogiques unipolaires, et d'un volt-mètre pour vérifier les tensions.

Croquis 06-es-analogiques-unipolaires (2/4)

Pour commander une sortie analogique en mode unipolaire, appeler `actionSortieAnalogique` :

```
actionSortieAnalogiqueUnipolaire (nomSortieAnalogique, valeur)
```

où :

- *nomSortieAnalogique* est `SORTIE_ANALOGIQUE_UNIPOLAIRE::SU0` ou `SORTIE_ANALOGIQUE_UNIPOLAIRE::SU1` ;
- *valeur* est un entier non signé de 32 bits (type `uint32_t`) :
 - entre 0 et 255, *valeur* spécifie la tension de sortie, avec la correspondance suivante : `tensionSortie = valeur * 41,412 mV` ;
 - au dessus de 255, il y a saturation, la tension de sortie est 10,56 V, c'est-à-dire la tension obtenue pour *valeur* = 255.

Croquis 06-es-analogiques-unipolaires (3/4)

Pour lire une entrée analogique unipolaire, appeler `lireEntreeAnalogiqueUnipolaire` :

```
lireEntreeAnalogiqueUnipolaire (nomEntreeAnalogique)
```

où :

- *nomEntreeAnalogique* est `ENTREE_ANALOGIQUE_UNIPOLAIRE::EU0` ou `ENTREE_ANALOGIQUE_UNIPOLAIRE::EU1`.

La fonction renvoie une valeur entière entre 0 et 1023 qui reflète la tension de l'entrée analogique suivant la relation : $valeurRenvoyée = tensionEntrée / 10,3226 \text{ mV}$.

Ainsi, pour une tension d'entrée de $10V$, on obtient théoriquement 969.

Croquis 07-es-analogiques-bipolaires (1/5)

Ce croquis montre comment commander les sorties analogiques bipolaires (c'est-à-dire la gamme [-10V, 10V]), et lire les entrées analogiques bipolaires (c'est-à-dire la gamme [-10V, 10V]).

Fonctionnement :

- l'encodeur fixe la tension des deux sorties analogiques (-128 -> -10,56 V, 127 -> 10,56 V) ;
- la valeur des sorties est affichée ;
- les entrées analogiques sont mesurées et affichées.

Pour vérifier le bon fonctionnement, il est utile de se munir de fils pour relier une sortie analogique bipolaire aux entrées analogiques bipolaires, et d'un volt-mètre pour vérifier les tensions.

Croquis 07-es-analogiques-bipolaires (2/5)

Pour commander une sortie analogique en mode bipolaire, appeler `actionSortieAnalogiqueBipolaire` :

```
actionSortieAnalogiqueBipolaire (nomSortieAnalogique, valeur)
```

où :

- *nomSortieAnalogique* est `SORTIE_ANALOGIQUE_BIPOLAIRE::SB0` ou `SORTIE_ANALOGIQUEBIPOLAIRE::SB1` ;
- *valeur*, est un entier signé de 32 bits (type `int32_t`) qui spécifie la tension de sortie :
 - entre -128 et 127 qui spécifie la tension de sortie, avec la correspondance suivante : $\text{tensionSortie} = \text{valeur} * 82,824 \text{ mV} - 10,56 \text{ V}$;
 - au dessus de 127, la tension de sortie est la même que la *valeur* = 127 ;
 - au dessous de -128, la tension de sortie est la même que la *valeur* = -128.

Croquis 07-es-analogiques-bipolaires (3/5)

Pour lire une entrée analogique en mode unipolaire, appeler `lireEntreeAnalogiqueBipolaire` :

`lireEntreeAnalogiqueBipolaire (nomEntreeAnalogique)`

où :

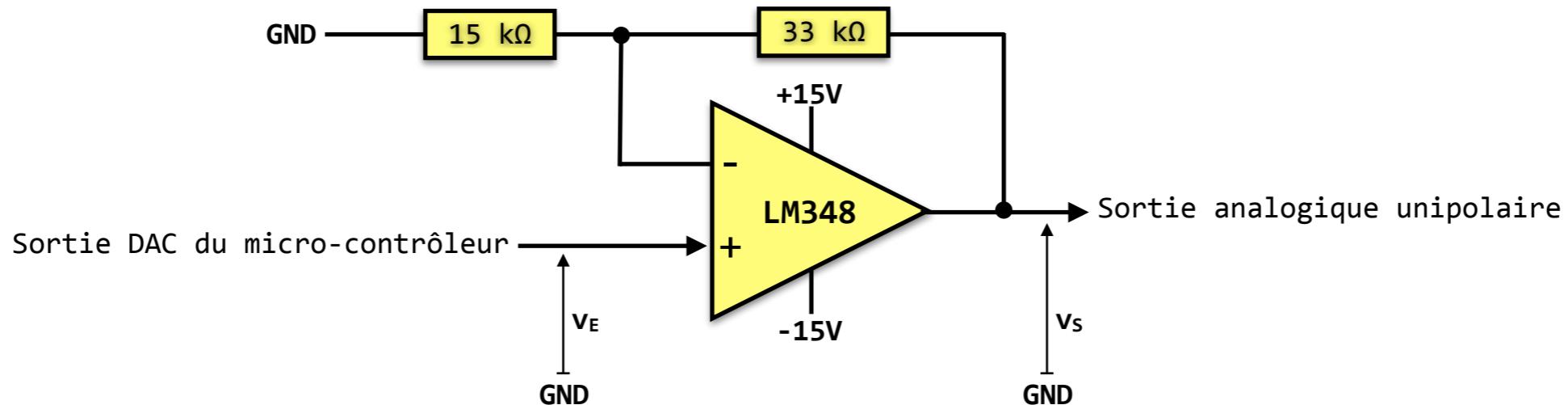
- *nomEntreeAnalogique* est `ENTREE_ANALOGIQUE_BIPOLAIRE::EB0` ou `ENTREE_ANALOGIQUE_BIPOLAIRE::EB1`.

La fonction renvoie une valeur entière entre -512 et 511 qui reflète la tension de l'entrée analogique suivant la relation : $valeurRenvoyée = tensionEntrée / 20,6452 \text{ mV}$.

Ainsi, pour une tension d'entrée de $10V$, on obtient théoriquement 969.

④ Schémas des E/S analogiques

Sortie unipolaire



La sortie DAC du micro-contrôleur fournit une tension de sortie v_E comprise entre 0V et 3,3V, en fonction de la valeur n (entre 0 et 255) du second argument de `actionSortieAnalogiqueUnipolaire`: $v_E = 3,3V * n / 255$.

Le montage de l'amplificateur opérationnel est du type *non inverseur*:

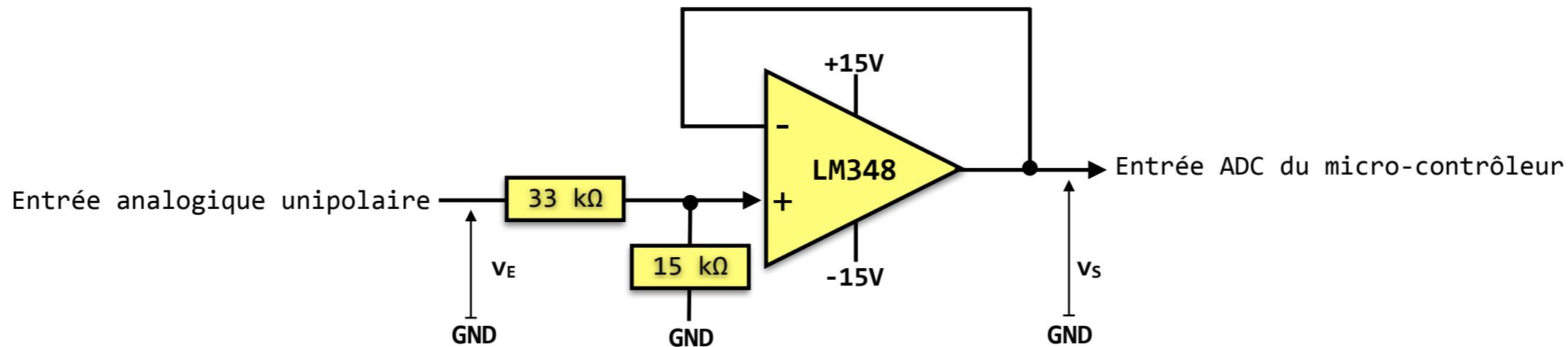
$$v_s = v_E * (15\text{ k}\Omega + 33\text{ k}\Omega) / 15\text{ k}\Omega$$

D'où :

$$v_s = 3,3V * n * (15\text{ k}\Omega + 33\text{ k}\Omega) / 15\text{ k}\Omega / 255 = n * 41,412\text{ mV}$$

Ainsi, pour $n=255$, on obtient $v_s=10,56\text{ V}$.

Entrée unipolaire



La fonction `lireEntreeAnalogiqueUnipolaire` convertit la tension v_s (comprise entre 0V et 3,3V) de l'entrée ADC du micro-contrôleur en une valeur entière n (entre 0 et 1023) : $n = v_s * 1023 / 3,3V$.

Le montage de l'amplificateur opérationnel est du type *suiveur*:

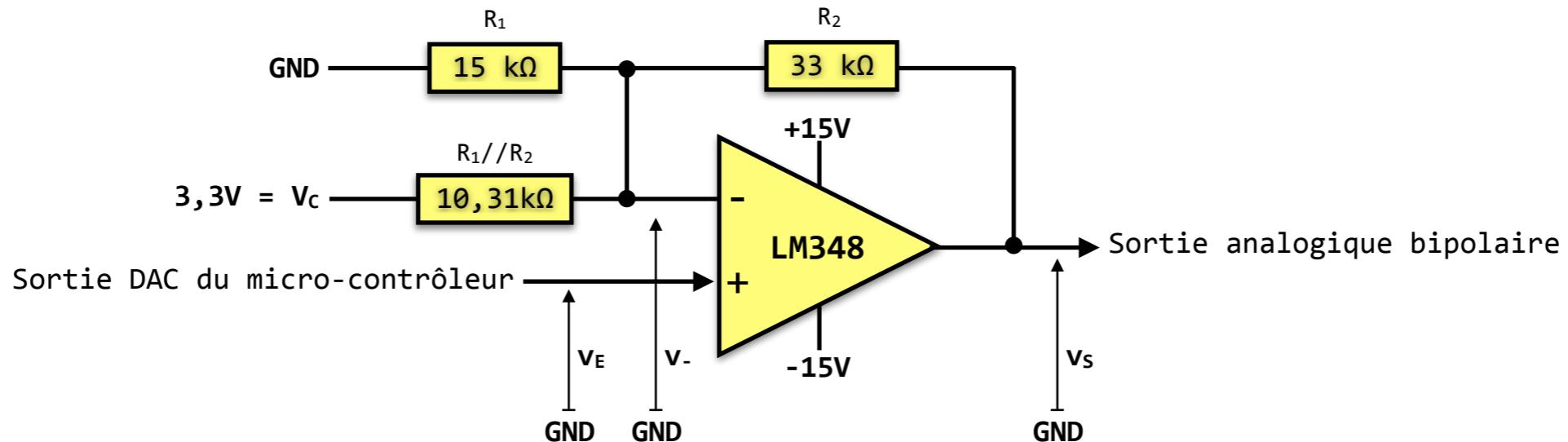
$$v_s = v_E * 15 \text{ k}\Omega / (15 \text{ k}\Omega + 33 \text{ k}\Omega)$$

D'où :

$$n = v_E * 1023 * 15 \text{ k}\Omega / (15 \text{ k}\Omega + 33 \text{ k}\Omega) / 3,3V = v_s / 10,3226 \text{ mV}$$

Ainsi, pour $v_E = 10V$, on obtient $n = 969$.

Sortie bipolaire



La sortie DAC du micro-contrôleur fournit une tension de sortie v_E comprise entre 0V et 3,3V, en fonction de la valeur n (entre 0 et 255) du second argument de `actionSortieAnalogiqueBipolaire` : $v_E = 3,3V * n / 255$.

Pour résoudre le montage, on écrit que la somme des courants arrivant à l'entrée inverseuse est nulle :

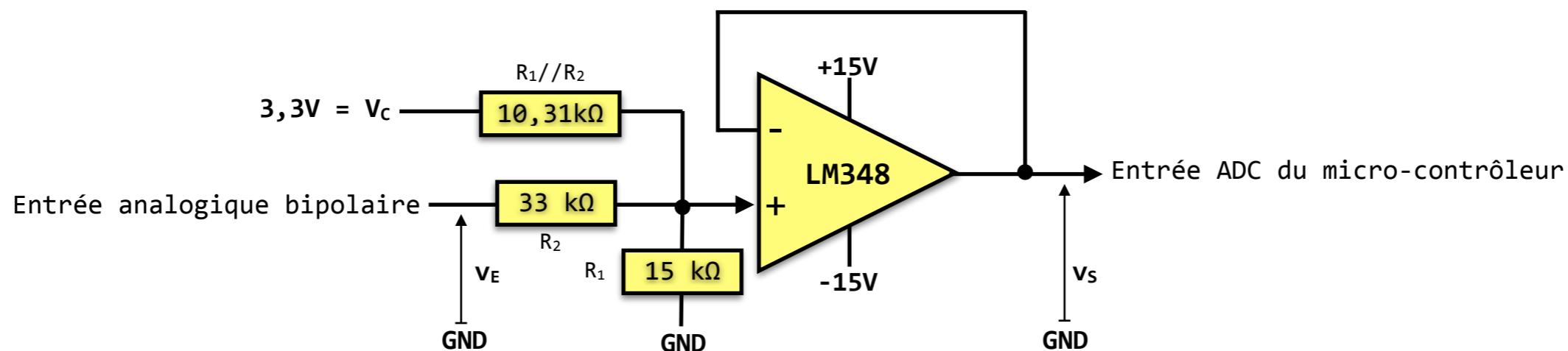
$$(v_s - v_-) / R_2 + (0 - v_-) / R_1 + (V_c - v_-) / (R_1 // R_2) = 0$$

D'où :

$$v_s = 2 * v_E * (R_1 + R_2) / R_1 - V_c * (R_1 + R_2) / R_1 = n * 82,824 \text{ mV} - 10,56 \text{ V}$$

Ainsi, $n=0 \rightarrow v_s=-10,56 \text{ V}$, $n=128 \rightarrow v_s=41 \text{ mV}$, $n=255 \rightarrow v_s=10,56 \text{ V}$.

Entrée bipolaire



La fonction `lireEntreeAnalogiqueBipolaire` convertit la tension v_s (comprise entre 0V et 3,3V) de l'entrée ADC du micro-contrôleur en une valeur entière n (entre 0 et 1023) : $n = v_s * 1023 / 3,3V$.

Pour résoudre le montage, on écrit que la somme des courants arrivant à l'entrée non-inverseuse est nulle :

$$(v_E - v_+) / R_2 + (0 - v_+) / R_1 + (V_c - v_+) / (R_1 // R_2) = 0$$

D'où :

$$v_s = v_E * R_1 / (R_1 + R_2) / 2 + V_c / 2$$

$$n = v_s / 20,6452 \text{ mV} + 511,5$$

Ainsi, $v_E = -10V \rightarrow n=27$, $v_E = 0V \rightarrow n=511$, $v_E = 10V \rightarrow n=996$.