TP Contrôle d’accès

Sommaire

TP contrôle d'accès

Introduction

Aujourd'hui, de nombreuses infrastructures telles que les établissements scolaires, les entreprises, les portails résidentiels ou encore les salles de sport sont équipées de systèmes de contrôle d'accès afin de sécuriser leurs entrées.
Dans ce cadre, notre objectif est de concevoir un **digicode** destiné à protéger l'accès à l'un de ces lieux.

Afin de mieux comprendre le fonctionnement du dispositif que nous allons réaliser, il est important de rappeler ce qu'est un digicode : il s'agit d'un dispositif électronique muni d'un clavier permettant de saisir un code alphanumérique, lequel déclenche l'ouverture d'une porte lorsque le code est reconnu comme valide.
Ce type de système est simple, efficace et largement utilisé, ce qui en fait un excellent support pour appliquer nos compétences en électronique et en programmation embarquée.

Matériels et Méthodologie

Matériels :

- Logiciel (Proteus, Arduino)
- Carte Arduino
- Un PCB
- Différents composant électronique

Méthodologie :

- Utilisation des compétences acquises pour concevoir le dispositif
- Rédaction d'un programme sur Arduino adapté aux besoins
- Simulation du fonctionnement via un jumeau numérique sur Proteus
- Réalisation du montage sur une carte réelle
- Validation de la carte avec le code Arduino

Conception du jumeau numérique

Composants électroniques

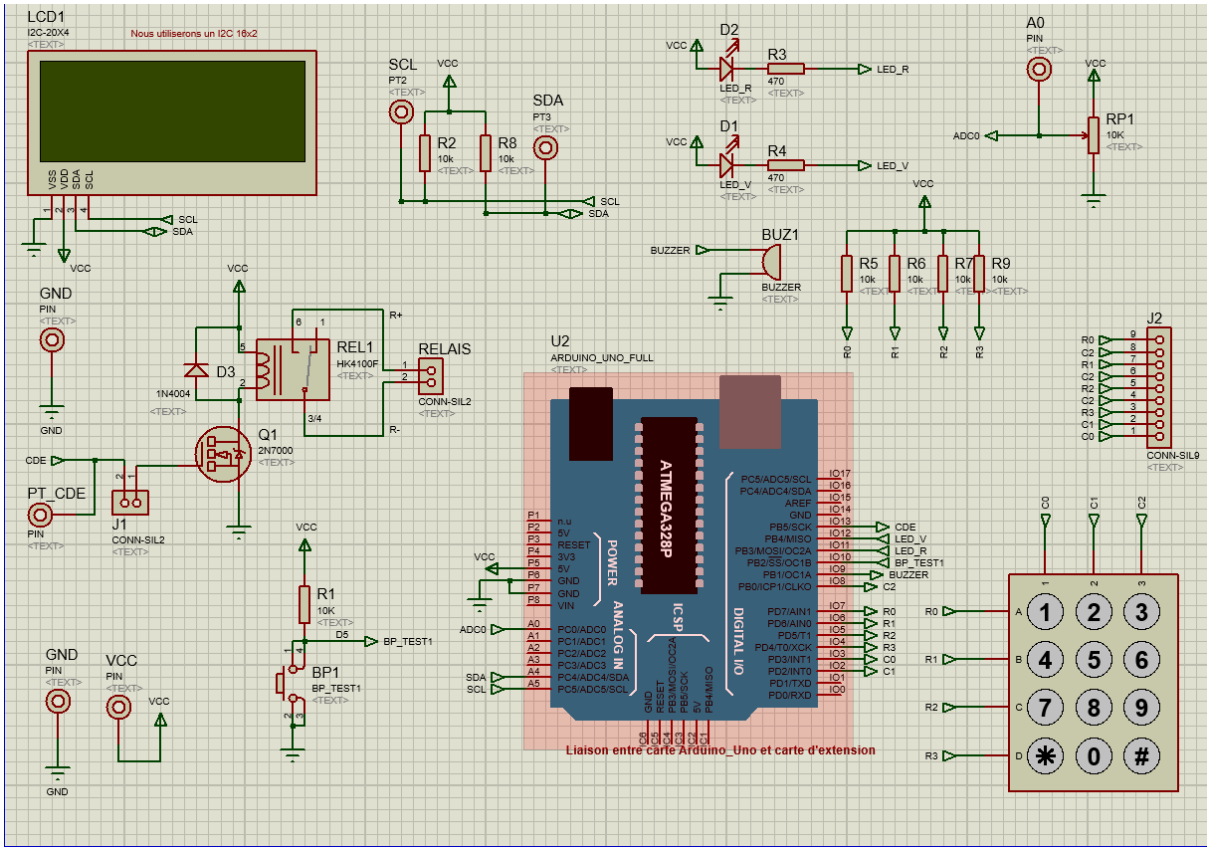
Pour concevoir le jumeau numérique d'un digicode, il est essentiel, dans un premier temps, d'identifier les différents composants électroniques entrant dans la composition d'un tel système.

Parmi les éléments les plus couramment utilisés, on retrouve :

- **Un clavier matriciel** (généralement 4x4 ou 4x3) : il permet à l'utilisateur de saisir le code d'accès. Chaque touche correspond à une combinaison de lignes et de colonnes que le microcontrôleur peut interpréter.
- **Un microcontrôleur** (comme une carte Arduino) : il reçoit les entrées du clavier, compare le code saisi avec celui enregistré en mémoire, et exécute les actions associées (ouverture, alerte, etc.).
- **Un écran (LCD ou OLED)** : il peut être utilisé pour afficher des messages à l'utilisateur (ex. : "Code accepté", "Erreur", "Entrée verrouillée").
- **Un relais ou un transistor de puissance** : il sert à activer le mécanisme de verrouillage (électroaimant, serrure électrique, etc.) lorsqu'un code valide est détecté.
- **Des LED et/ou un buzzer** : utilisés comme indicateurs visuels et sonores pour signaler les actions (confirmation, erreur, verrouillage temporaire).

L'ensemble de ces composants sera modélisé dans le logiciel de simulation Proteus, afin de réaliser un jumeau numérique fidèle du dispositif réel. Ce modèle virtuel permettra de tester le comportement du digicode, d'optimiser le code et de valider le fonctionnement du système avant toute mise en œuvre physique.

Intégration dans l'environnement de simulation Proteus



L’ensemble des composants décrits précédemment a été intégré dans un environnement de simulation à l’aide du logiciel Proteus. Le schéma ci-dessus représente le jumeau numérique du digicode que nous avons conçu.

Ce schéma comporte notamment :

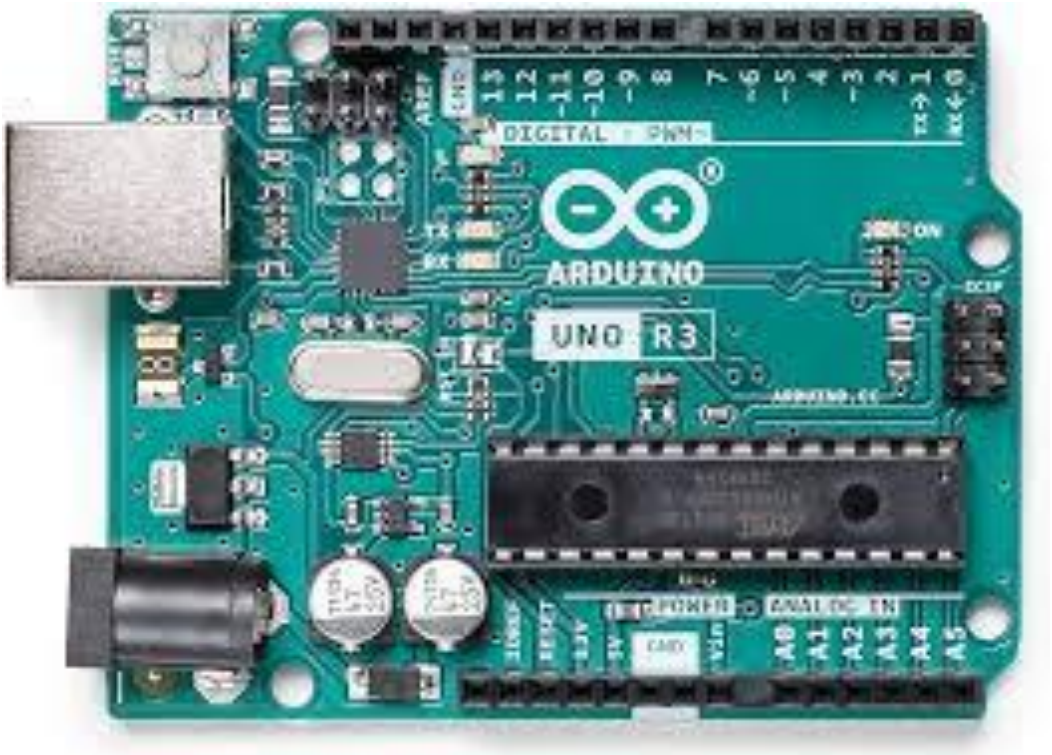
- **Un clavier matriciel 4x3** connecté aux broches numériques de l’Arduino pour la saisie du code.
- **Une carte Arduino UNO (ATmega328P)** utilisée comme microcontrôleur principal, en charge de la lecture des entrées, du traitement du code et du pilotage des sorties.
- **Un écran LCD 16x2 avec interface I2C**, facilitant la communication avec l’Arduino et permettant l'affichage de messages d’état ou d’instructions.
- **Un relais commandé par un transistor MOSFET (2N7000)** pour simuler l’ouverture de la porte en cas de code correct. Une diode de roue libre (1N4004) est également ajoutée pour protéger le circuit contre les surtensions.

- **Un buzzer (BUZ1)** utilisé comme alarme sonore en cas d'erreur ou de tentative d’accès non autorisé.
- **Deux LED (verte et rouge)** permettant d’indiquer visuellement l’état du système (accès autorisé ou refusé).
- **Des résistances de pull-down et pull-up** pour stabiliser certaines entrées.

Ce jumeau numérique permet de tester le comportement complet du digicode, du moment où l’utilisateur saisit un code, jusqu’à l’ouverture ou le refus d’accès, en passant par les signaux visuels et sonores. Il offre un environnement de test sûr et flexible, avant toute réalisation physique sur PCB.

Élaboration du programme

Pour pouvoir commencer la simulation il nous faut tout d’abord élaborer un programme pour chaque situation. La rédaction du programme se réalisera sur Arduino et sera implémenté sur le jumeau numérique pour la simulation et ensuite une carte Arduino Uno pour la réalisation.



Écran et clavier

Pour pouvoir utiliser un écran et un clavier il faut en premier temps installer les bibliothèques <LiquidCrystal_I2C.h> et <Keypad.h>

Ensuite il faut définir l'adresse de l'I2C avec ses dimensions

LiquidCrystal_I2C lcd(0x3F, 16, 2);

```
// Définir les broches du clavier matriciel
const byte ROWS = 4; // Quatre lignes
const byte COLS = 3; // Trois colonnes
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};
byte rowPins[ROWS] = {7, 6, 5, 4};
byte colPins[COLS] = {3, 2, 8};
// Créer l'objet Keypad
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);
```

Pour créer le clavier il faut créer des variables avec les dimensions de ce dernier (4*3) et un tableau de tout ce qu'il contient comme touche. "rowPins" et "colPins" sont là pour faire les connexions entre les broches de l'Arduino et le clavier.

```
if (customKey) {
  Serial.print("Touche pressee: ");
  Serial.println(customKey);

  lcd.setCursor(0, 1); // Positionner le curseur en haut à gauche
  lcd.print("Touche: ");
  lcd.print(customKey);
  lcd.print("    "); // Effacer les anciens caractères en ajoutant des espaces
}
```

Dans la boucle principale on cherche à récolter l'information du clavier, dès qu'une touche est entrée elle sera alors affichée sur l'écran.

L'instruction customKey revoie la valeur de la touche frappée.

Gestion du mot de passe

Pour cette partie nous avons choisi un mot de passe (3630). L'objectif est de pouvoir récolter les informations du clavier afin de les comparées avec une variable qui contiendra nôtre mot de passe.

L'ajout d'un mot de passe nous complexifie à cause de la gestion des boutons # et *. Ainsi nous avons cherché une utilisation de ces 2 boutons, le bouton # sert à valider et le bouton * sert à annuler le mot de passe que l'on est en train de taper.

```
// Gérer les entrées du clavier
if (customKey == '*') {
  // Réinitialiser la saisie
  inputPassword = "";
  lcd.clear();
  lcd.print("Annuler");
  delay(100);
  lcd.clear();
```

Ici on détecte si la touche * est pressé, si elle les on peut donc effacer le code rempli.

```
else if (customKey == '#') {
  // Valider la saisie
  if (inputPassword == password) {
    lcd.clear();
    lcd.print("Mot de passe OK");
    inputPassword = "";
  } else {
    lcd.clear();
    lcd.print("Mot de passe NON");
    inputPassword = "";
  }
```

La touche # nous sert de validation, une fois pressé il faut vérifier si le code est bon puis écrire un message en fonction de ce résultat.

```
const String password = "3630";
String inputPassword = "";
```

Le mot de passe que l'on souhaite rentrer est une variable vide où l'on ajoute à chaque lecture de clavier la touche choisie à la fin de cette variable.

Le code final

Pour réaliser le programme final, il nous faut tout d'abord reprendre le principe d'un digicode où l'on ajoute une petite fonction d'initialisation.

Ainsi en premier temps il nous faudra appuyer sur un bouton qui donneras plusieurs instructions puis, après avoir taper le bon code la gâche qui sert à la fermeture du dispositif s'ouvrira.

En premier nous avons lu les données de la broche 10 (le bouton) qui s'il se trouve à l'état "LOW" envoie les instructions de base

```
if (a == LOW)
{
  lcd.clear();
  lcd.print("* = annuler");
  lcd.setCursor(0, 1);
  lcd.print("# = valider");
  delay(200);
  lcd.clear();
  lcd.print("Rentrez");
  lcd.setCursor(0, 1);
  lcd.print("mot de passe");
  delay(200);
  lcd.clear();
  valeur = 1;
}
```

La variable valeur nous sert de cadenas, elle permet de bloquer la suite du programme pour ne pas commettre d'erreur en cas d'appuie accidentel ou trop rapide. Pour la suite on a repris la partie précédente, mot de passe où l'on vérifie en plus si valeur est à 1 (bouton a été appuyer). Après le déroulement du programme ne change en rien jusqu'à ce que le mot de passe remplie soit bon.

Ajout de la gâche sur la broche 13 de l'Arduino

```
// Définir la gache
const int gache = 13;
```

Le système d'ouverture d'une porte n'est pas infini, nous avons donc rajouter un délai pour refermer la gâche

```
if (inputPassword == password) {
  lcd.clear();
  lcd.print("Mot de passe OK");
  inputPassword = "";
  digitalWrite(gache, HIGH);
  delay(2000);
  digitalWrite(gache,LOW);
  valeur = 0;
```

On aurait très bien pu penser que la gâche se referme toute seul en claquant la porte mais dans un espace de simulation un délai de 2s est suffisant pour comprendre l'ouverture.

La simulation sur proteus

Le jumeau numérique que l'on à créer au tout début était entier, ici on va prendre pour chaque partie de programme les parties qui nous intéresse du jumeau.

Le but d'isolé et de ne pas tout tester d'un seul coup est de mieux cerné les erreurs. Si l'on ne teste que le clavier et l'écran une erreur est plus facile à prédire et à réparer.

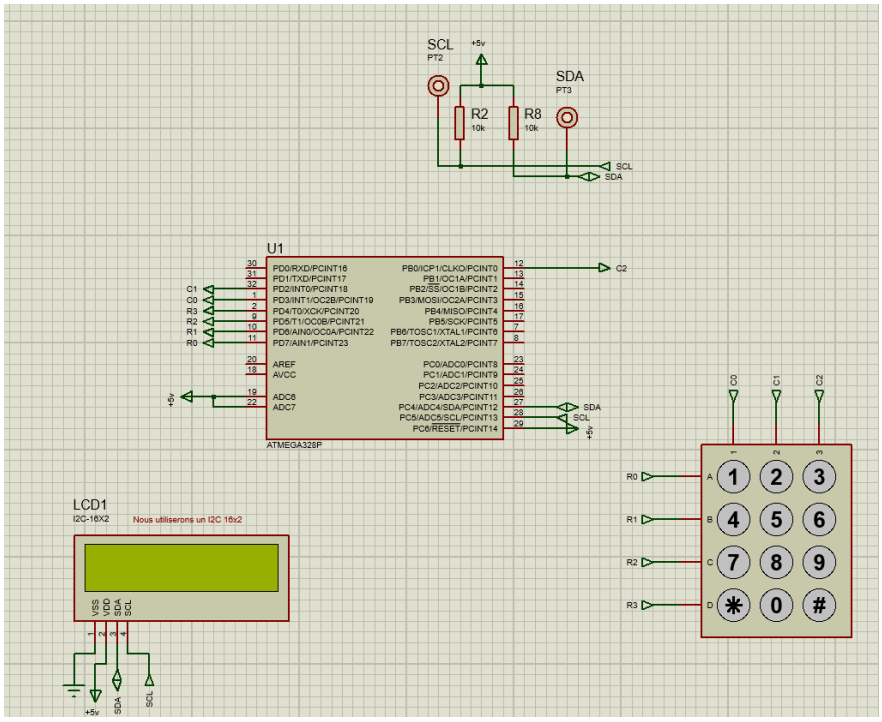
Test écran et clavier

Ainsi pour commencer on a séparé le schéma de base pour n'obtenir que la partie écran et la partie clavier. Ce test va surtout nous intéresser pour comprendre les interactions entre les broches de l'Arduino nôtre programme et les différentes connexions.

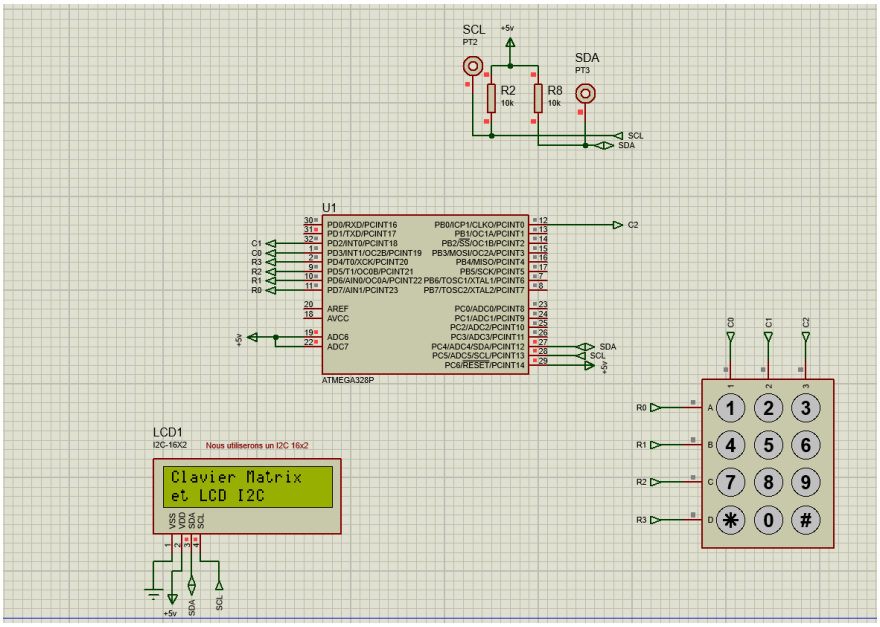
Clavier --> Arduino

Arduino --> écran

Ci-dessous la partie pour simuler le clavier et l'écran seulement.

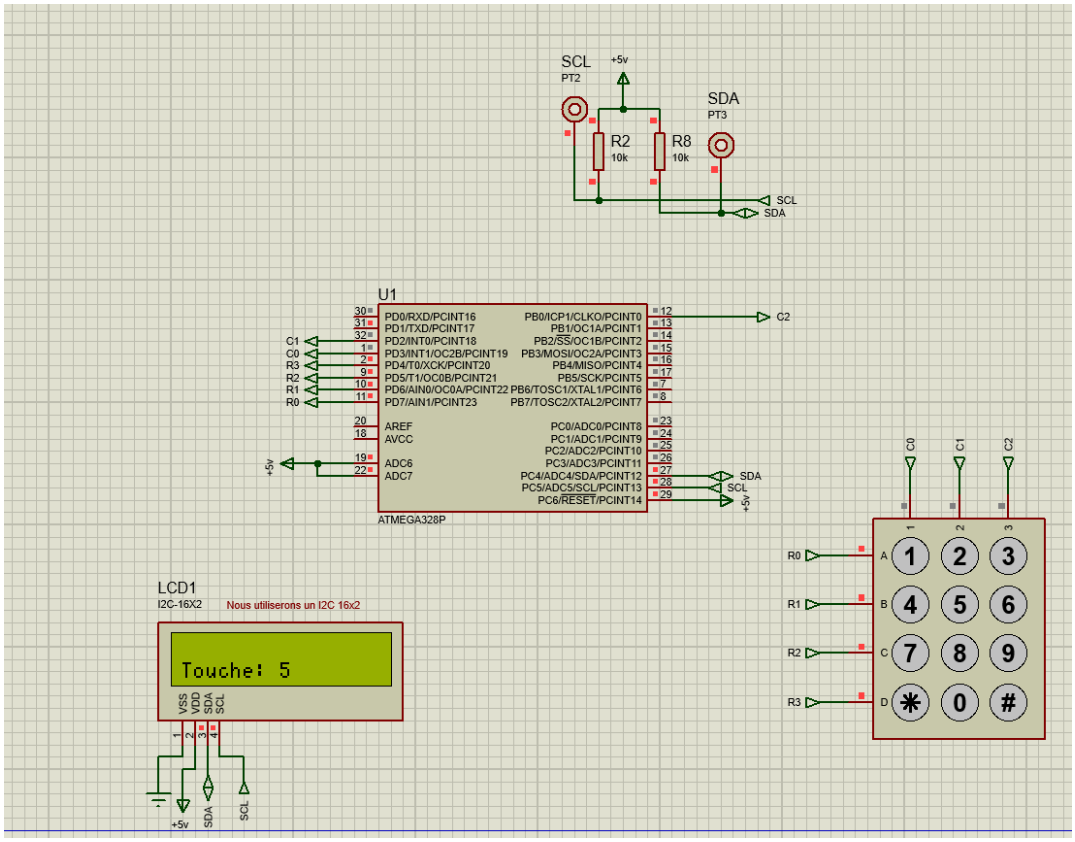


Une fois après avoir implémenté le programme et lancé la simulation, un affichage d'initialisation de l'écran devais s'afficher. Les simulations sur proteus étant lente il faut attendre pour chacune d'entre elle une quinzaine de secondes avant que cela ne commence.

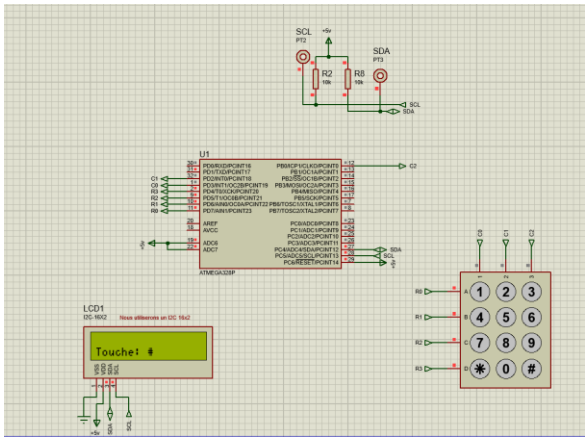


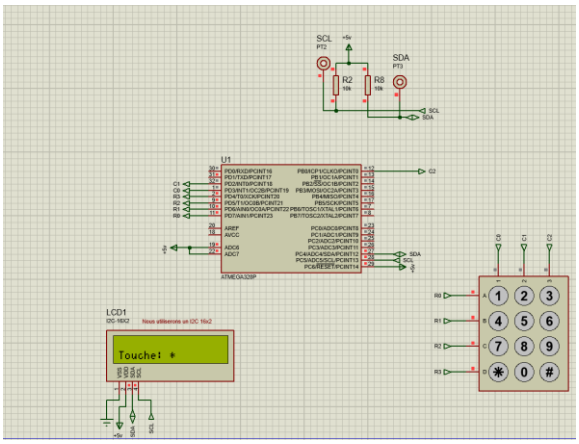
Ici l'initialisation de l'écran fonctionne bel et bien. Dans le programme il nous reste plus que la partie clavier à tester.

Ensuite il nous faut rentrer une touche et elle est sensé s'afficher sur l'écran.



Ici on peut observer que la touche 5 une fois appuyé est affiché sur l'écran. Cependant on rencontre un problème puisque nous avons seulement les touches qui s'affichent.





Ainsi quand les 2 touches * et # sont rentrer rien ne se passe à part l'affichage.

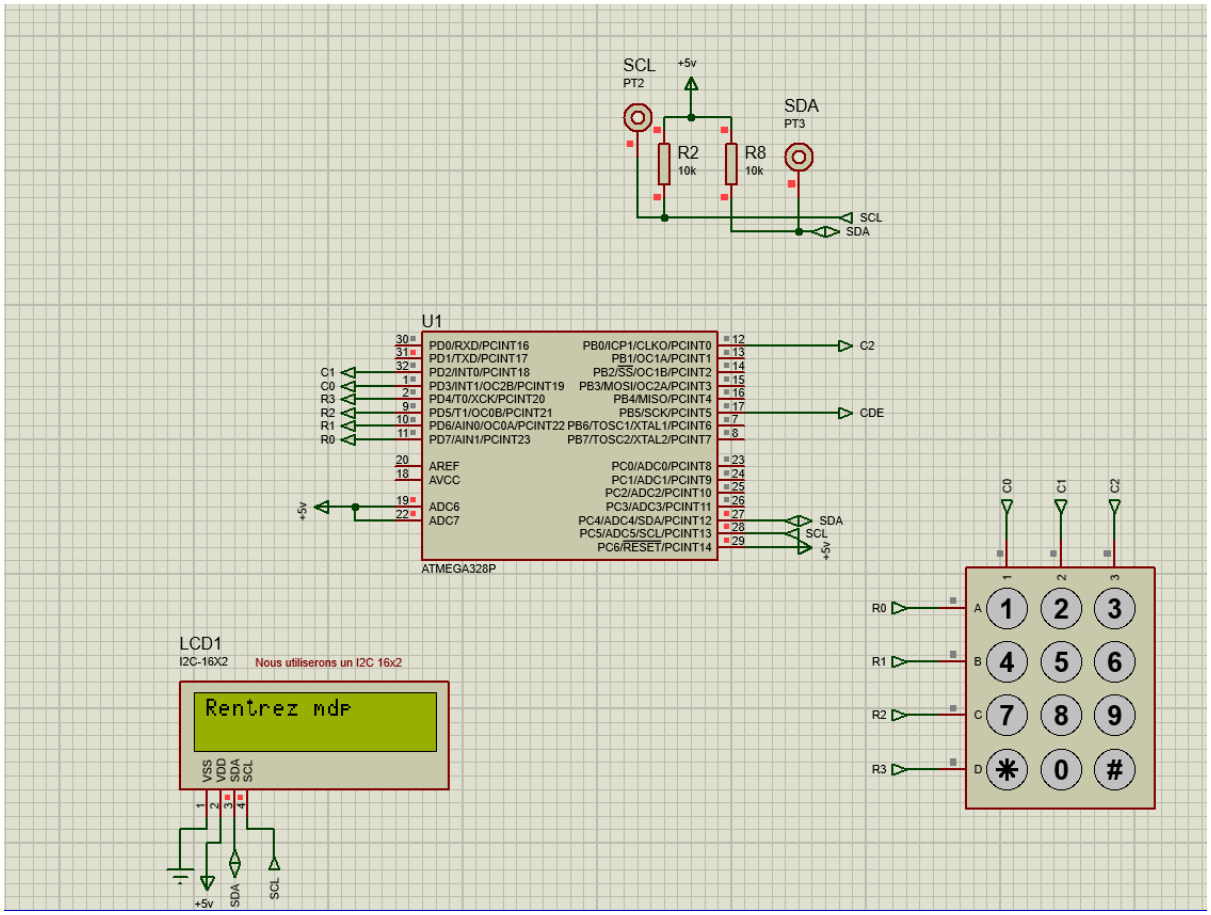
L'affichage et le clavier étant simuler et correcte il faut maintenant prendre en compte de ce qui est taper.

Test mot de passe

Le jumeau numérique ne change pas pour cette partie, seul le programme nous permet la gestion de collecte des données du clavier.

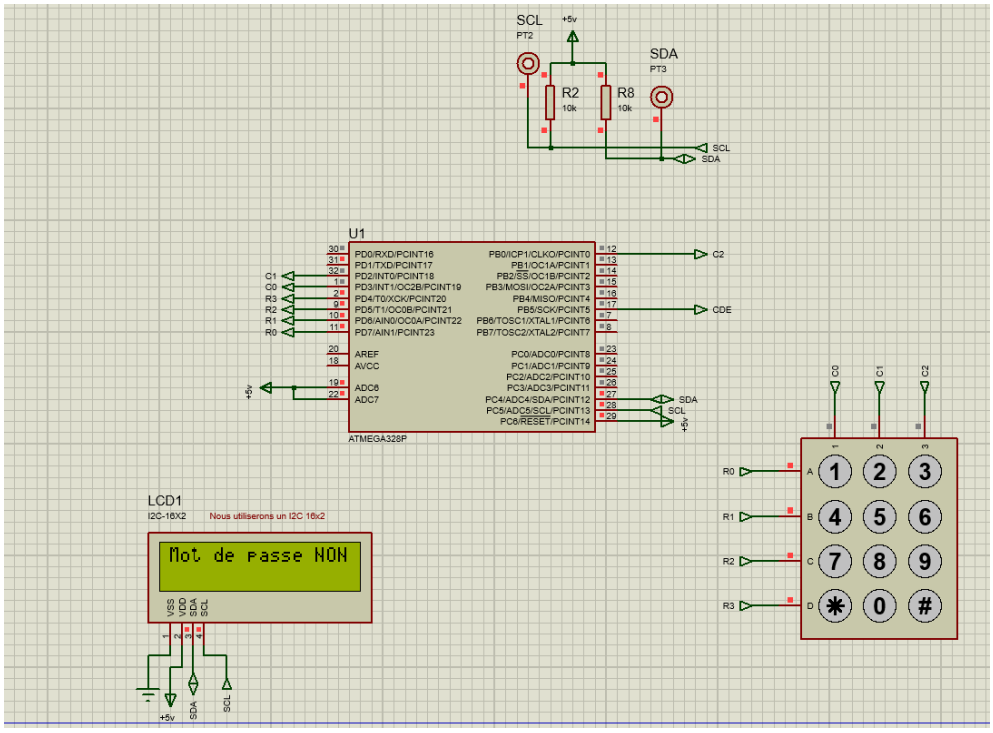
Comme dit précédemment la gestion du clavier comportait le problème des touches * et # qui nous servent donc à réinitialiser et valider le mot de passe.

Ici nous commençons avec une initialisation qui nous dit de rentrer un mot de passe

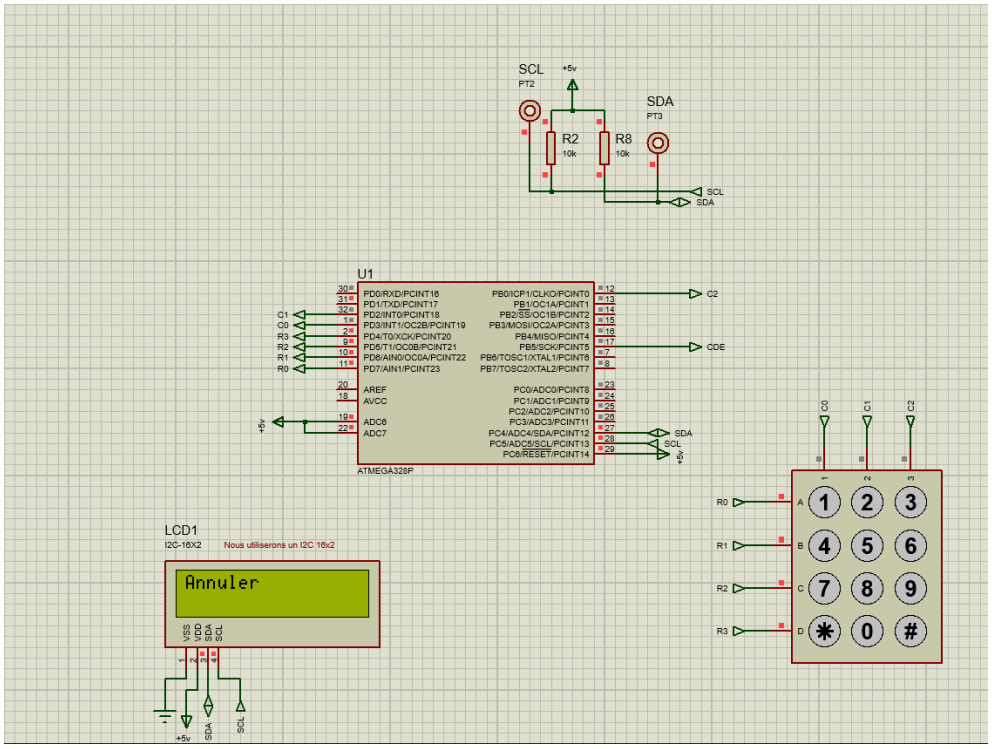


Le seul problème qui peut potentiellement arriver est que l'on puisse taper une touche du clavier avant que le message soit fini. Ce problème n'est jamais sensé arrivé puisque le message se trouve dans la boucle setup.

Nous avons cherché à écrire un mot de passe incorrect (différent de 3630) pour voir si la partie programme était réussi et ne validai pas n'importe quoi.

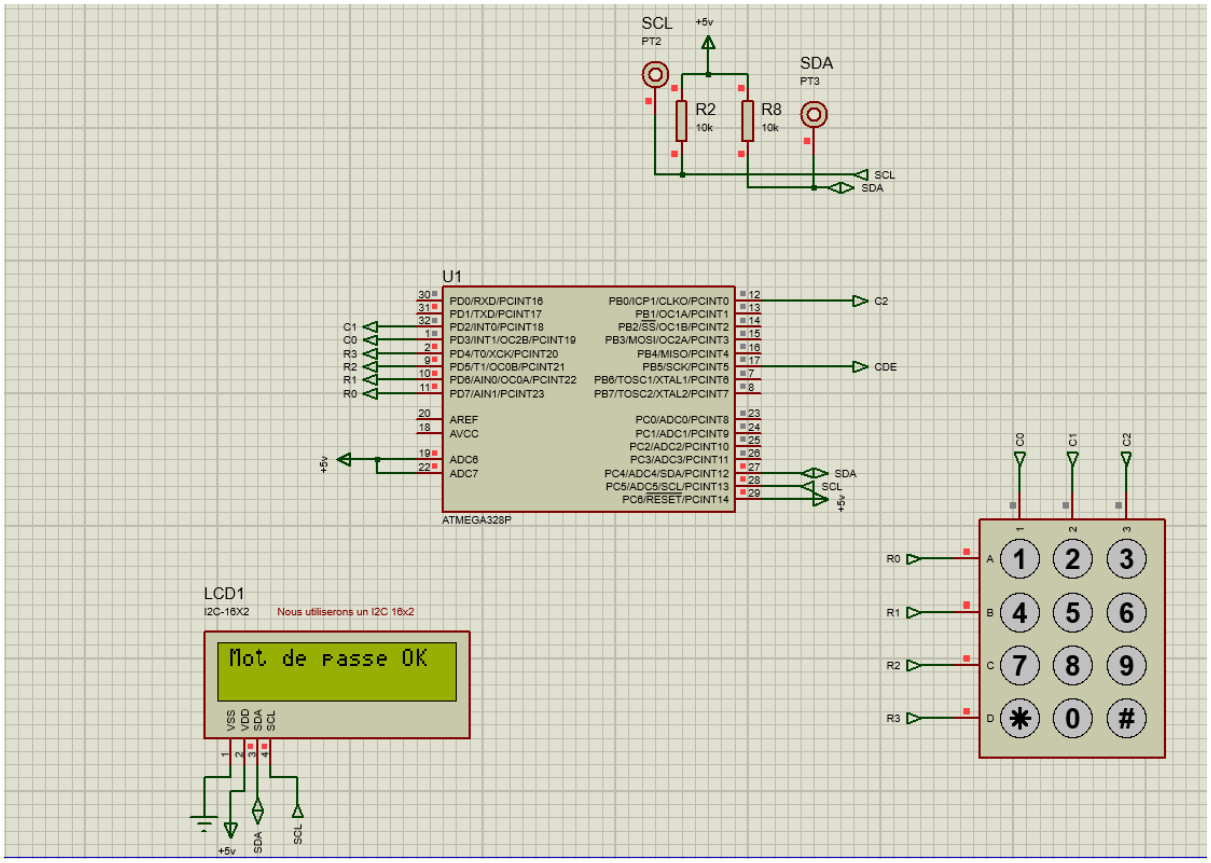


Ensuite il nous faut tester la fonction qui réinitialise tout le mot de passe pour que l'on puisse en taper un nouveau.



Enfin il nous reste plus qu’à rentrer le bon mot de passe (3630).

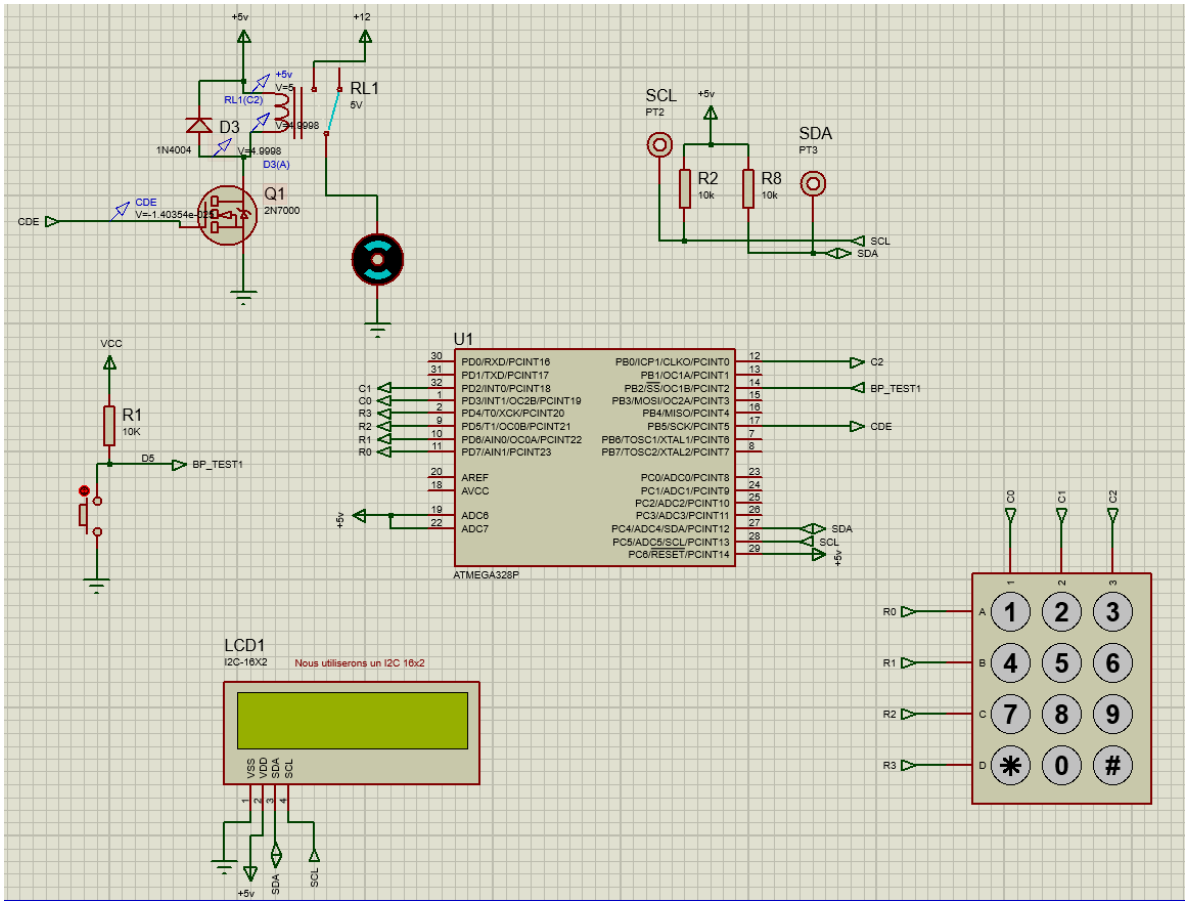
Ceci va nous permettre de vérifier plusieurs choses comme si le mot de passe s’efface bien, mais surtout savoir s’il s’enregistre bien dans la variable pour comparer.

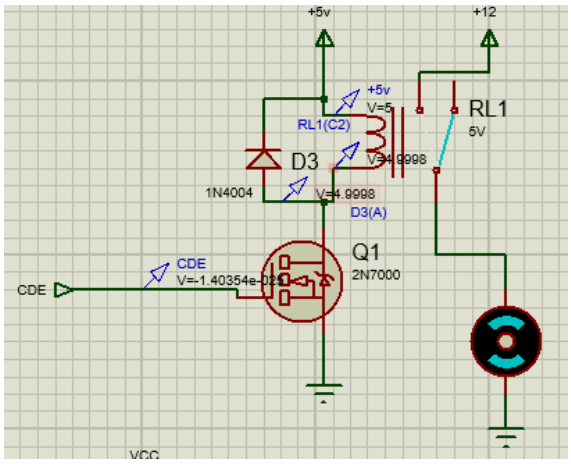


La gestion des informations transmis par le clavier sont donc corrects. Maintenant il nous reste plus qu'à savoir quoi en faire de ces informations.

Test final

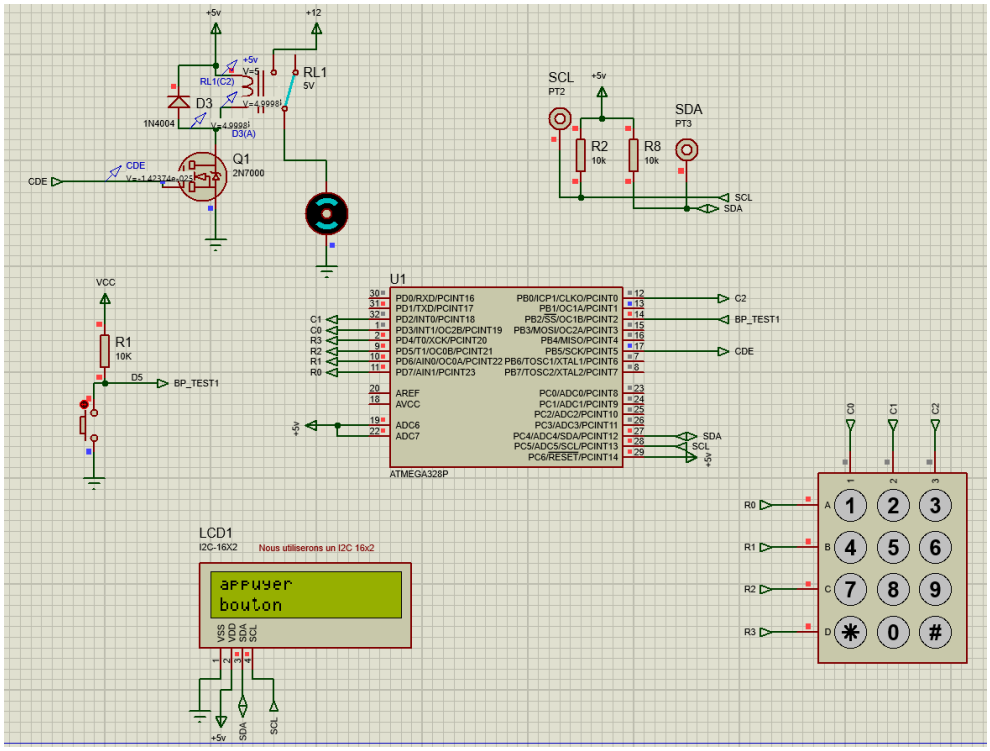
Pour rappel le digicode doit pouvoir être allumer via un bouton et une gâche doit s'ouvrir quand le code est bon. Il y a donc un ajout de blocs sur le jumeau numérique, une section bouton et une autre avec un moteur.





Le moteur ici simule la gâche, si le code est bon le moteur tourne.

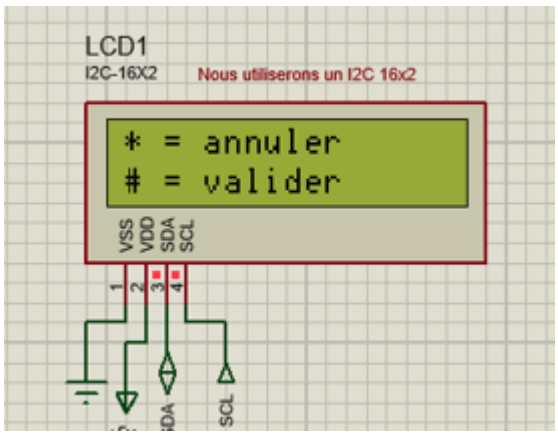
Avec l'ajout du bouton il y a donc un message d'initialisation dans la boucle setup.



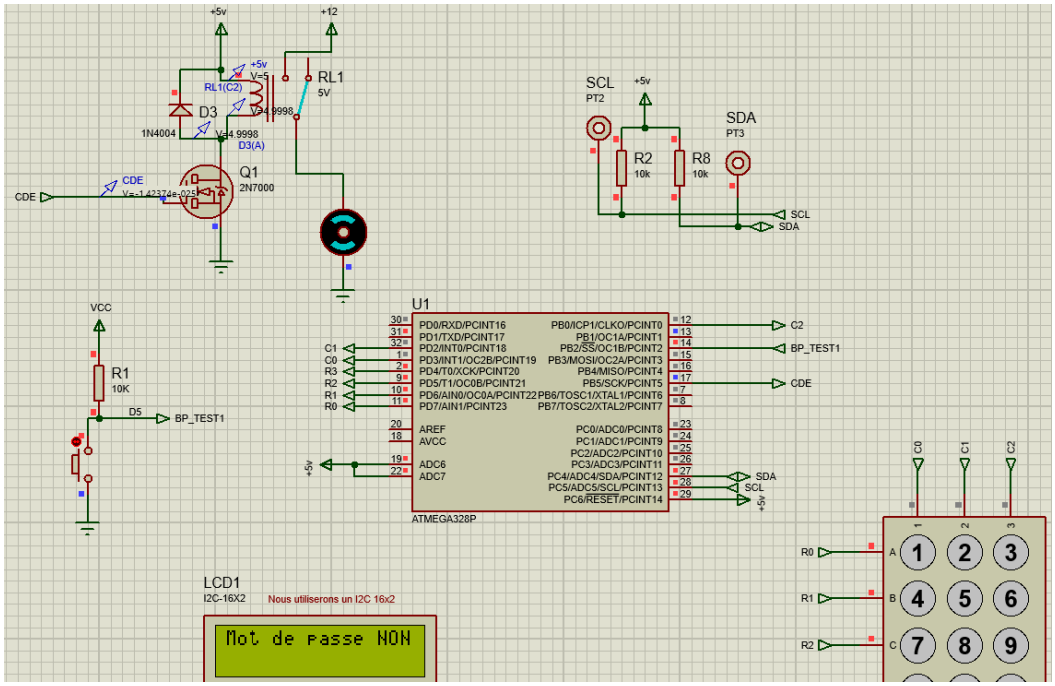
Pour éviter les problèmes liés au clavier, un ajout d'un cadenas dans le programme bloque l'utilisation de ce dernier temps que le bouton n'est pas appuyé.

Il n'y ait pas sensé il y avoir ce problème mais il vaut mieux sécuriser en prévision.

Nous avons aussi rajouté un affichage avant "rentrer mdp" qui nous donne quelques petites instructions sur l'utilisation des touches * et #

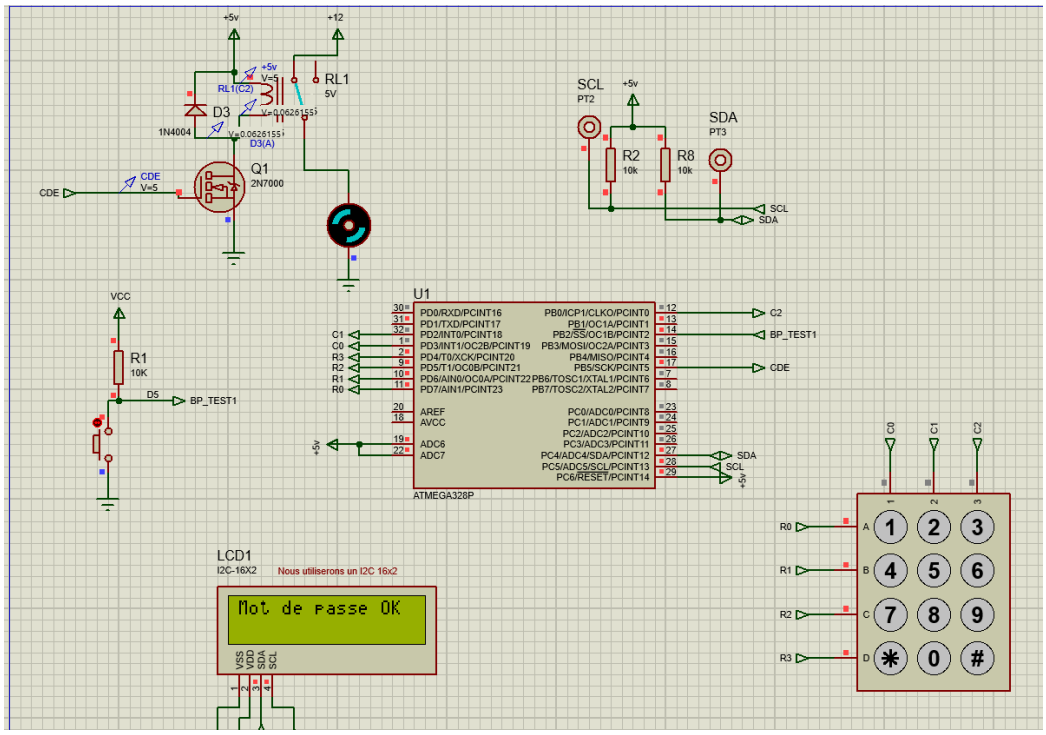


Maintenant il nous faut vérifier si quand la touche # est appuyé (même si le mot de passe est faux) le moteur ne tourne pas.



Le mot de passe rentrer est incorrect on peut observer que le moteur ne bouge pas car le relai n’a pas basculé sur l’entrer +12v.

Il ne reste donc plus qu’à vérifier le relai quand le mot de passe est correct.



Grâce à ce dernier test on peut donc affirmer que le programme final est réussi.

Les problèmes rencontrés

La partie jumeau numérique est validé cependant elle nous a susciter plusieurs problèmes majeurs.

Le problème principal était la connexion d'une broche du clavier (C2) à la broche Tx de l'Arduino.

```
byte colPins[COLS] = {3, 2, 8};
```

C'est ainsi que l'on connecte C2 à la broche 8 de l'Arduino qui n'était pas utilisé.

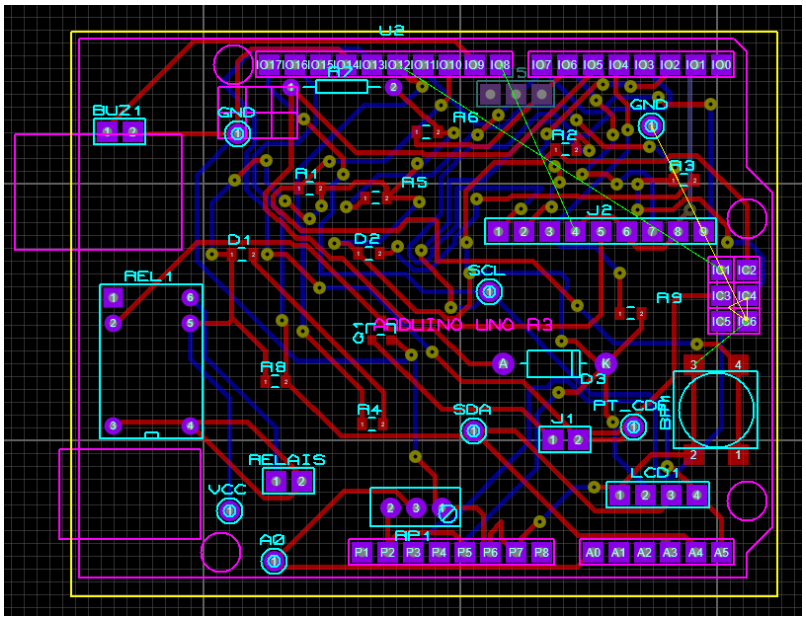
L'autre grosse erreur était la non-suppression du mot de passe quand il était incorrect. Si un mdp était rentrer, faux, on écrivait à la suite sans effacer la variable, ce qui nous donnait un mot de passe avec trop de caractères.

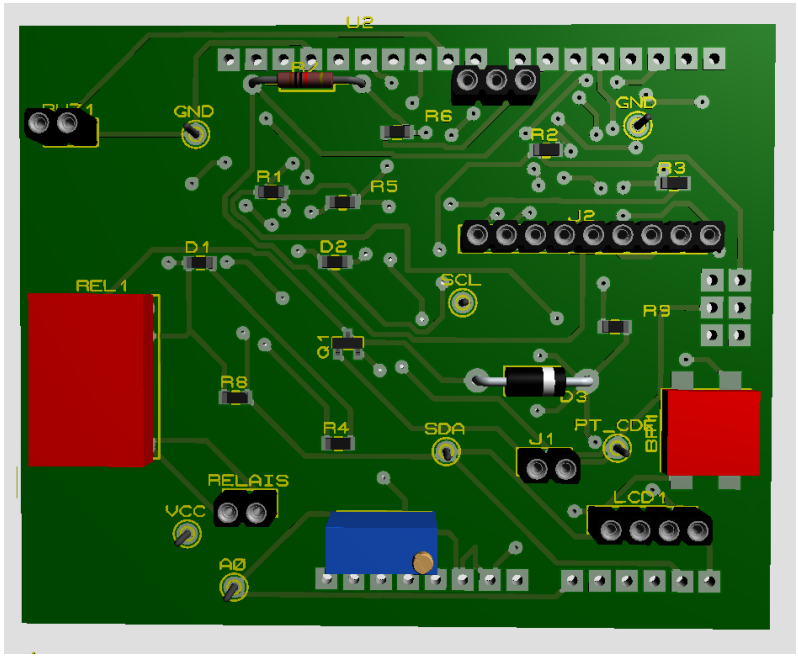
Le reste des problèmes étaient mineurs, mais surtout plus simple à détecter. Les petits défauts étaient des délais (trop long ou trop court) et des affichages pas très clair.

Réalisation de la carte

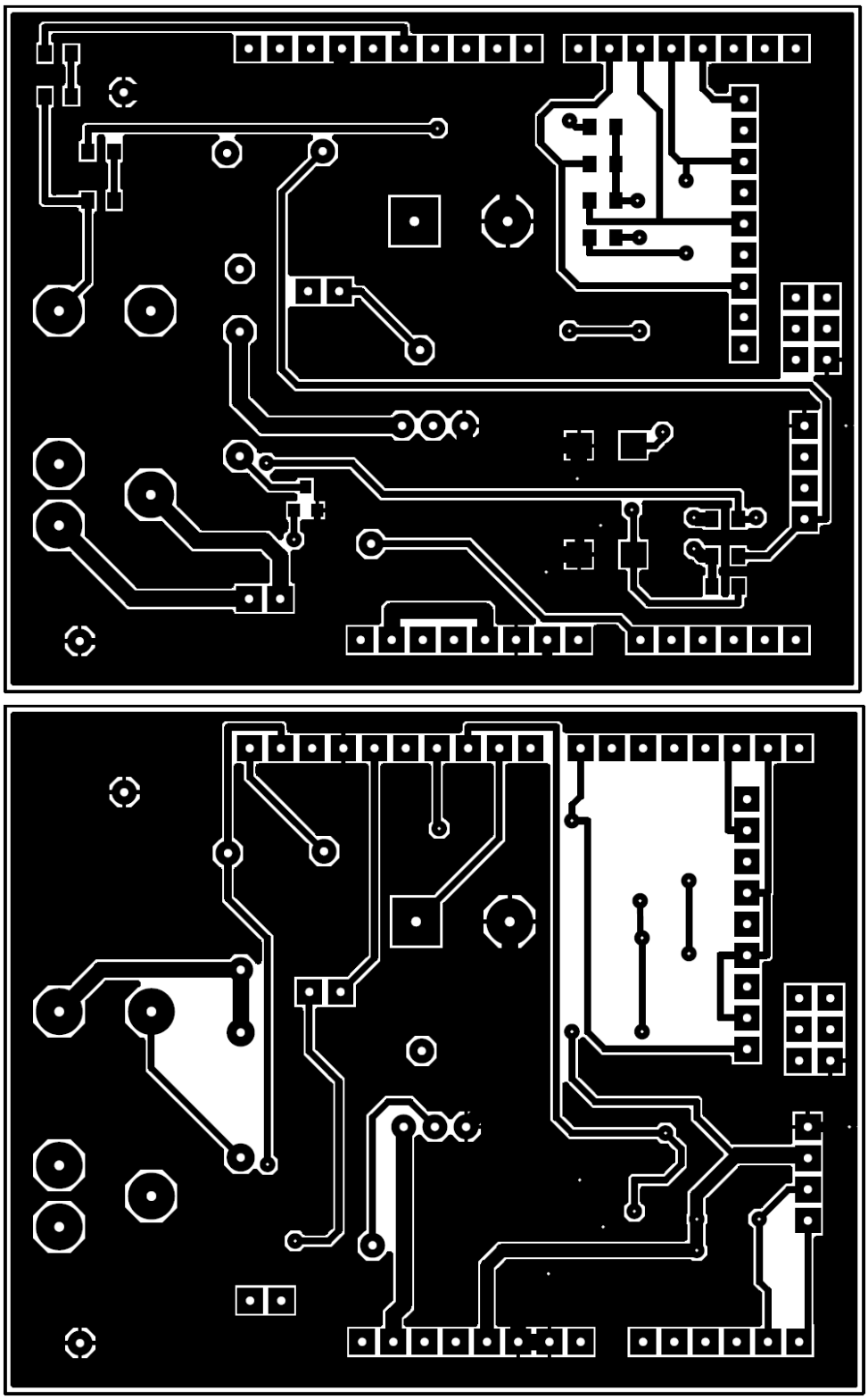
Soudure / Brasure

Maintenant que nôtre jumeau numérique fonctionne correctement, il nous faut à présent réaliser la soudure des composants monté en surface (CMS). Grâce à la réalisation du schéma sur Proteus nous avons pu obtenir une représentation 3d et un schéma quasiment représentatif de la carte finale.

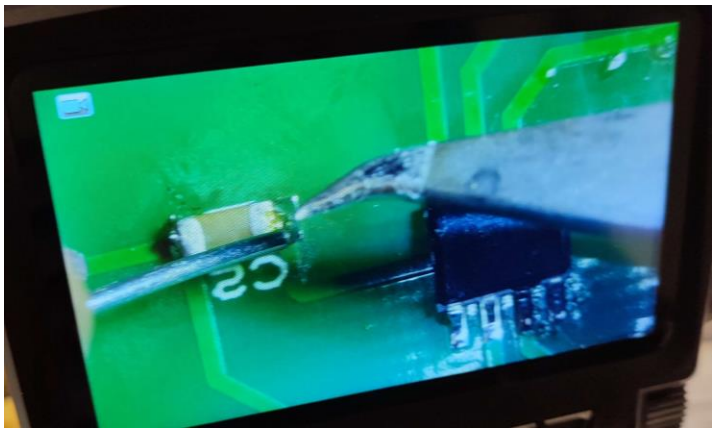




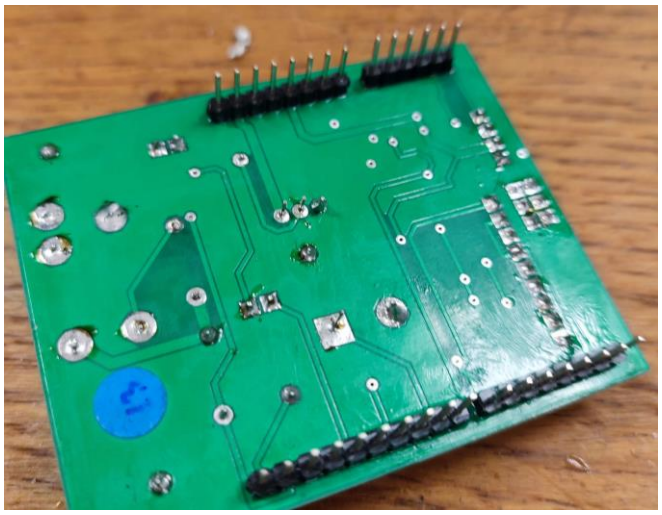
Pour réaliser nos soudures nous allons plus tôt utiliser ce schéma qui nous offre un meilleur visuel finalement et une lecture plus simple différentes connexions.



L’utilisation d’un matériel adapté pour les brasures est utilisée. Sur cette image on peut observer un condensateur zoomé au microscope pour faciliter l’action.



Une fois avoir terminé les soudures, nous pouvions commencer les tests avec le programme Arduino.



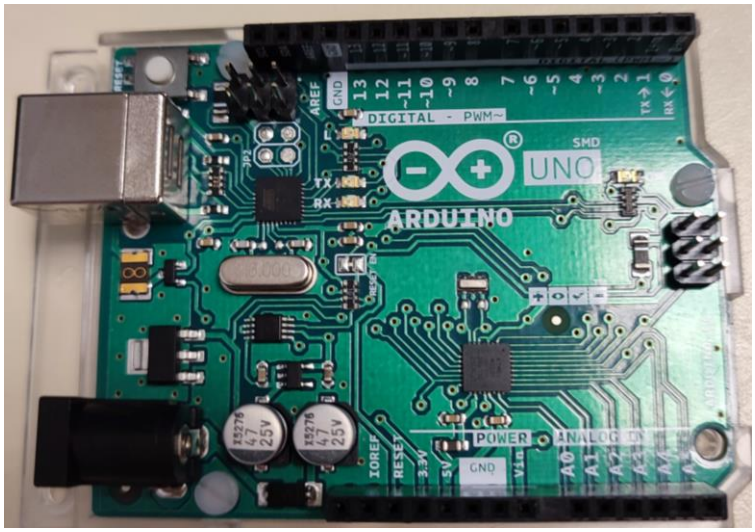
Sur le dessous de la carte un fil à wrappé a été rajouté afin de corriger un problème de conception (pas présent sur cette photo).

En électronique, le **wrapping** est une technique alternative de réalisation de circuit imprimé, consistant à remplacer les pistes gravées et les soudures par des fils enroulés sur les broches des composants, pour produire des circuits complexes en petit nombre.

Le problème que nous avons eu pendant la simulation était aussi présent sur le PCB, pour pallier celui-ci nous avons coupé à l'aide d'un cutter la piste que reliai la broche TX de l'Arduino et la broche 3 du clavier. Ensuite à l'aide d'un multimètre nous avons vérifié si les broches étaient encore reliées et enfin, nous avons mis le fil à wrapper pour relier le clavier à une nouvelle broche de l'Arduino non utilisé.

Simulation de la carte

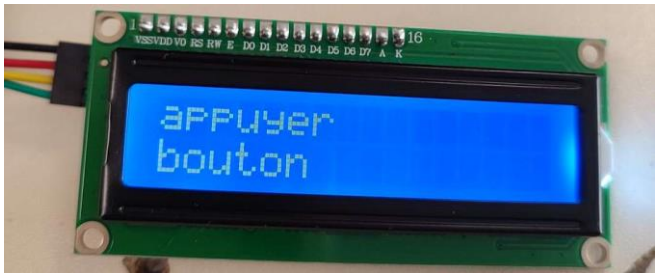
Nôtre carte était théoriquement terminé, il nous restait plus qu'à tester si elle fonctionnait correctement grâce au programme Arduino.



Puisque le jumeau numérique était correct nous pouvons juste tester le programme final qui ne comporte que deux parties interactives avec la carte, le clavier et l'écran.

Pour l'affichage de l'écran seul la ligne ci-dessous a été modifié.

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```



Ainsi l'affichage sur l'écran I2C est vérifié et correcte par rapport aux attendus.

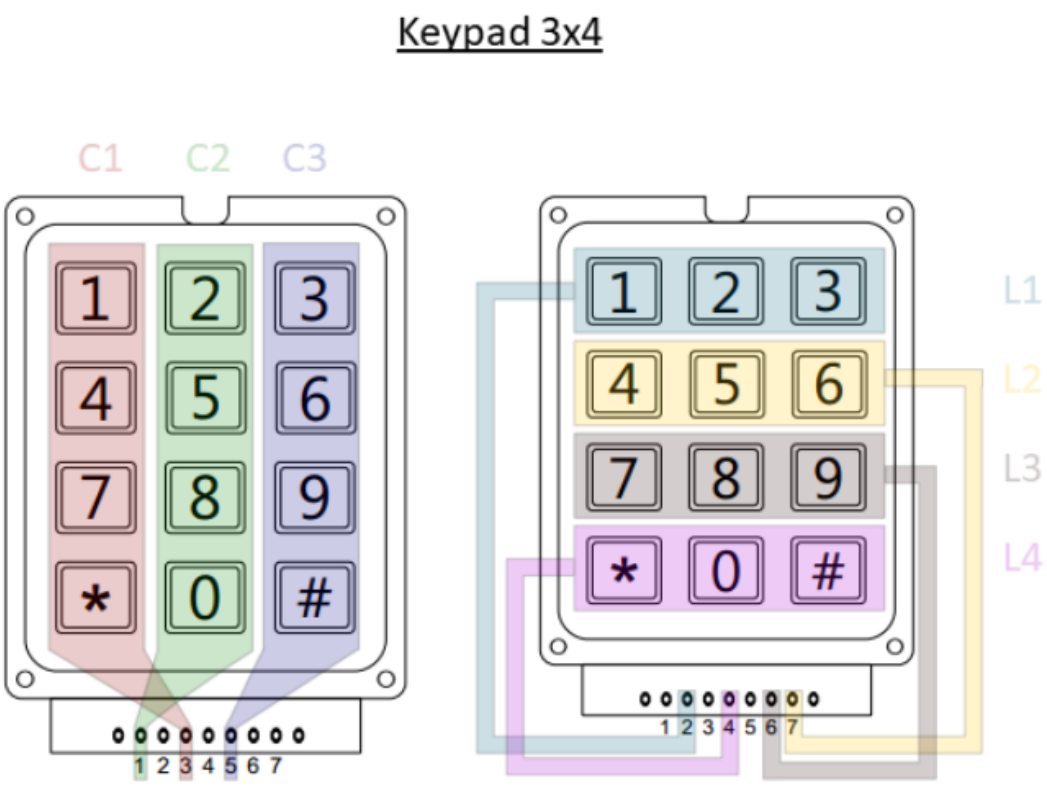
Pour vérifier si le clavier répondait correctement il nous suffisait d'appuyé sur une touche pour voir l'affichage. Ici la touche * a été pressé.

Un nouveau problème apparait. Les broches du clavier ne correspondent pas.

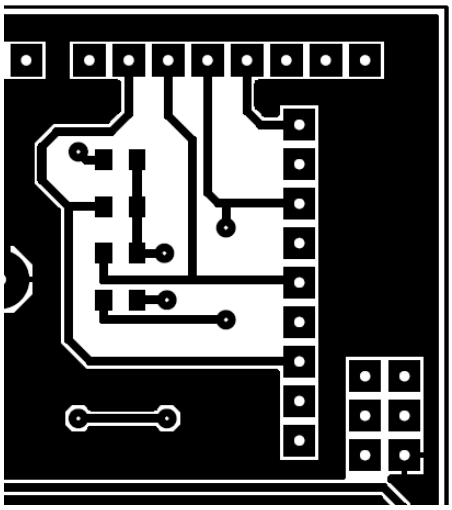
Réparation de la carte

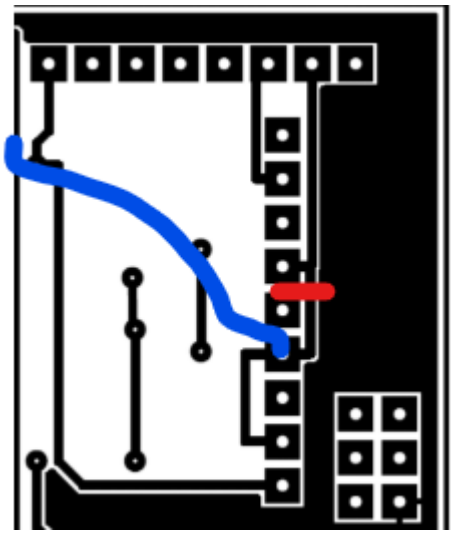
Nous avons un problème à cause du clavier qui n'est pas connecté aux bonnes broches de l'Arduino. Dans un premier cadre de recherche nous avons réeffectuer les soudures du clavier afin de vérifier si ce n'était pas une simple erreur.

Après avoir resouder la partie clavier --> broche Arduino et n'ayant obtenu aucun résultat, nous avons analysé le comportement du clavier (tel touche fait fonctionner tel broche).



Avec l'aide du schéma de soudure nous avions pu voir sur quelle broche était connecté celle du clavier.





Cependant sur le schéma vu de dessous (deuxième image) on peut voir que 3 broches du clavier sont connectées entre elle. Le trait rouge est la piste coupée et le bleu est le fil à wrappé qui est relié à la broche 8.

On peut donc enfin observer le problème final il y a une broche du clavier qui est encore connecté à la broche Tx de l'Arduino, et une autre qui est connecté à celle où l'on a mis un fil à wrapper.

Grâce au schéma du comportement du clavier on apprends que les broches 0 et 8 du clavier ne sont pas utilisé, on peut donc brancher nos 2 autres broches dessus afin de les connectés correctement.

Voici une image de la carte réparer avec l'ajout de 2 autres files à wrapper et la coupure des 2 piste gênante.

Une fois avoir modifié la gestion des broches il faut modifier dans le programme les lignes et les colonnes où se connecte le clavier.

On passe donc de :

```
byte rowPins[ROWS] = {7, 6, 5, 4};
byte colPins[COLS] = {3, 2, 8};
```

À :

```
byte rowPins[ROWS] = {6, 2, 4, 5};
byte colPins[COLS] = {8, 7, 3};
```

Par rapport à l'image du comportement du clavier il nous suffit d'observer où chaque broche est connecté pour modifier le programme.

Le test final

Toutes les broches sont connectées séparément les unes des autres, le programme est modifié en fonction des dernières modifications, donc il ne reste plus qu'à retester le digicode.

Pour tester le digicode une vidéo a été réaliser, les images suivantes proviennent de cette dernière.

La gestion du mot de passe annuler avec la touche * est bien pris en compte, la vidéo à été réaliser avec les tests de tous les principaux boutons d'affiler. Il n'y a pas de réinitialisation.

La gestion de validation si le mot de passe est aussi correct.

On peut supposer que le bouton annuler et l'option le mot de passe est faux fonctionne, puisque la dernière option (le mot de passe est correct) fonctionne. Les 2 premières possibilités effacent le mot de passe ainsi il y a juste à valider avec le bon code d'accès pour valider les 2 autre.

Ici le mot de passe est correct, cependant il est impossible de le voir en image car un ajout de son a été réaliser. Nous avons rencontré une dernière contrainte qui elle est physique. Normalement quand la touche # (valider) est appuyé une LED verte ou une rouge est sensé s'éclairé. Le problème étant le clavier qui vient se poser sur la carte et cache la visibilité des LED, alors nous avons décidé de rajouter un son avec un Buzzer (1 bip mot de passe correct 2 bip mot de passe faux) pour confirmer l'ouverture ou non de la porte.