الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Mention Électronique
Spécialité Traitement de l'Information et Systèmes Électroniques

présenté par

BENARFA MALIKA

&

GHODBANE KHADIDJA

Implémentation d'un CODEC Hamming sur ARDUINO MEGA

Proposé par : Mr Anou. A

Année Universitaire 2015-2016

Remerciements

A l'issue de cette phase de nos études, atteinte grâce à Allah Le Tout-Puissant qui nous a donné santé, la volonté et patience, nous tenons à adresser nos plus vifs remerciements à notre promoteur, Monsieur *Anou Abderrahmane* qui nous a encadré, suivi et encouragé.

Nous remercions aussi vivement tous les membres du jury de bien vouloir juger ce travail.

Nos remerciements vont également à tous nos enseignants du Département d'électronique.

Les discussions, les remarques et les commentaires de Mr *KAZAD* ont été source d'idées et ont contribué au développement et à l'amélioration de ces études.

De même que nous ne pouvons exclure de ces remerciements toutes les personnes qui nous ont aidés de près ou de loin dans la réalisation de ce mémoire.

Sans oublier de remercier tout particulièrement nos parents pour leur soutien inconditionnel tout au long de nos longues années d'études.

ملخص:

لمعالجة البيانات وتحسين موثوقية نقلها، والذي من الضروري تأمينه، نستخدم الرموز الصحيحة. هذا هو الهدف من هذه المذكرة التي تعرف بنظام ترميز Hammingعلى بطاقة Arduino Mega . يرتكزعملنا بشكل رئيسي على أربعة محاور. وهكذا يخصص الفصل الأول لعموميات عن نظرية المعلومات وكذلك مصدر المبرمج والقناة. ونتطرق إلى مختلف رموز اكتشاف وتصحيح الأخطاء في الفصل الثاني حيث سيتم وصف كل نوع مع مختلف المصفوفات المعتمدة. أما في الفصل الثالث نتعرف على Arduino ومبدأ تشغيلها، ومختلف أنواعها وخصائصها. ويشمل الفصل الرابع التطبيق الخاص بدراسة مختلف أجزاء النظام، مع تسليط الضوء على بطاقة متحكم Arduino، رسم بياني لها، وpinout من شاشة LCD، والترميز، ومراقبتها، متلازمة وتهيئة ترميزات من كلمة 8 بت والترميز التناظري.

كلمات المفاتيح: المعالجة، ARDUINO، متلازمة، كشف وتصحيح، قناة

Résumé : Pour traiter l'information et améliorer la fiabilité de sa transmission qu'il est nécessaire de sécuriser, on utilise des codes correcteurs. C'est l'objet de ce mémoire qui présente un système d'un codec Hamming sur l'ARDUINO MEGA.

Notre travail s'articule principalement sur quatre axes. Ainsi, le premier chapitre est consacré à des généralités sur la théorie de l'information ainsi que le codeur source et canal. Les différents codes de détection et correction des erreurs sont abordés, quant à eux, dans le second chapitre où sera décrit chaque type avec les différentes matrices adoptées. Le troisième portera sur l'initiation à ARDUINO, on y définit le principe de fonctionnement des cartes ARDUINO, leurs différents types et leurs caractéristiques. Le quatrième chapitre comportera l'application concernant l'étude des différentes parties de ce système, en mettant en évidence la carte ARDUINO à microcontrôleur, son schéma fonctionnel, son brochage de l'afficheur LCD, le codage, le contrôle de la partie, le syndrome et l'initialisation des codages d'un mot de 8 bits ainsi que le codage analogique.

Mots clés : ARDUINO MEGA, code en bloc, codage de Hamming, détection d'erreur, correction d'erreur, syndrome.

Abstract :

For a data processing and to improve the reliability of his transmission, which it is necessary to make safe, one, uses correct codes. It is the object of this memory, which presents a system of a codec Hamming on the ARDUINO. Our work is articulated mainly on four axes. Thus, the first chapter is devoted to general information on the information theory as well as the coder source and channel. The various codes of detection and correction of the errors are approached, as for them, in the second chapter where each type with the various adopted matrices will be described. The third will relate to initiation with ARDUINO, one defines in it the principle of operation of charts ARDUINO, their various types and their characteristics. The fourth chapter will comprise the application concerning the study of the various parts of this system, by underlining chart ARDUINO to microcontroller, his functional diagram; it is stitching of billposter LCD, coding, and the control of the part, the syndrome and the initialization of codings of a word of 8 bits as well as analog coding.

Keywords : Hamming, ARDUINO, syndrome, detection and correction, channel

Listes des acronymes et abréviations

ADC	:	Analog to Digital Converter
ARQ	:	Automatic Repeat Request
B	:	Bruit
BPSK	:	Binary Phase-Shift Keying
C1	:	Condensateur
CAN	:	Convertisseur analogique-numérique
CSR	:	Convolutif Systématique Récursifs
d_H	:	Distance de Hamming
d_{min}	:	Distance minimal
E_b	:	erreur par bit
EPROM	:	Electrically-Erasable Programmable Read-Only Memory ou mémoire morte Effaçable électriquement et programmable
E_s	:	Erreur par symbol
FEC	:	Forward Error Correction
G	:	Matrice génératrice
H	:	matrice de contrôle de parité
H^T	:	Matrice de contrôle de parité transposée
ICSP	:	In-Circuit Serial Programmation
I_k	:	La matrice identité
LCD	:	Liquid Crystal Display ou afficheur à cristaux liquide
LED	:	Luminance éléctrique diode
MIC		Modulation par impulsion code
N_0	:	Bruit initial
NSC	:	Code convolutifs non systématique
PIC	:	Programmable Interface Controller
PSK	:	Phase-Shift Keying
PWM	:	Pulse-Width Modulation
QPSK	:	Quadratic Phase-Shift Keying
R	:	le rendement binaire.
R1	:	Résistance
RISC	:	Reduced Instructions Set Computer
S	:	Signal
S	:	Syndrome
SNR	:	rapport signal à bruit
SPI	:	interface serie périphérique
SPI	:	Interface Série Périphérique
T	:	Pouvoir de correction
t'	:	pouvoir de détection
TEB	:	Taux d'erreur binaire
TWI	:	Two Wire Interface
VDD	:	Voltage Drain Drain
VSS	:	Voltage Source Source
$W_{(x)}$	:	Le poids de Hamming

SOMMAIRE

Liste des figures

Liste des tableaux

Introduction générale

Le principe du codage correcteur d'erreurs (ou codage canal) est d'ajouter aux données à transmettre, des données redondantes de manière à rendre plus fiable la transmission. Ce principe peut être couplé avec le codage de source qui consiste à rendre l'information transmise la plus concise possible sans dégrader les performances du système. Généralement, le codage de source sert à compenser la redondance introduite par le codage canal. L'étude présentée dans ce mémoire se limite au codage canal.

Un exemple simple de codage canal consiste à doubler l'information, c'est-à-dire à envoyer deux fois les données à émettre, de sorte que si une partie des données est mal détectée dans la première séquence, elle puisse être récupérée dans la deuxième. Pour les codes utilisés en pratique, chaque donnée redondante est calculée en fonction de chacune des données de l'information utile. Cela permet, en plus d'apporter de la redondance aux informations, de les sécuriser puisqu'il est nécessaire en réception de connaître la règle pour décoder l'information reçue.

Les informations traitées au niveau du codage sont sous forme de symboles binaires, (0 ou 1) ou de blocs de symboles binaires formant ainsi des mots.

Les paramètres caractérisant un code sont sa taille, ainsi que sa redondance. La taille correspond au nombre de bits (ou de mots) émis, y compris ceux de redondance. La redondance est exprimée en pourcentage et correspond au nombre de bits de redondance sur le nombre de bits utiles émis. Un code C de taille N ayant N- K bits (ou mots) de redondance est noté C (N,K). La performance des codes en termes de correction d'erreurs dépend de K, de la taille N du code et de la méthode de construction.

Pour une méthode de construction donnée et une longueur N fixée, plus il y a de redondance, plus le pouvoir de correction est élevé. Cependant une redondance élevée entraîne une diminution du débit utile. Ainsi, il y a nécessairement un compromis entre pouvoir de correction et débit utile.

Les codes correcteurs sont classifiés en différentes familles ou classes de codes, en fonction de la manière dont la redondance est calculée et insérée dans les données utiles.

On distingue deux catégories, les codes en blocs et les codes convolutifs dont les principales caractéristiques du premier sont présentées dans les chapitres suivants. L'application du codage canal aux transmissions. Dans ce mémoire, un seul type de codes en blocs est étudié en détails, est le code de Hamming, qui sera implémenté dans deux cartes ARDUINO MEGA. la première carte est utilisée en émission pour l'implémentation du codeur de Hamming, la seconde est utilisée en réception pour l'implémentation du décodeur de Hamming.

Chapitre 1 **Généralités du codage canal**

1.1. Introduction

La communication à distance (téléphone, télévision, satellite, etc.) entre machines et usagers nécessite des lignes de transmission acheminant l'information sans la modifier. Les systèmes de transmission numérique (figure 1.1) utilisés véhiculent de l'information entre une source et un destinataire en utilisant un support physique comme le câble, la fibre optique ou, encore, la propagation sur un canal radioélectrique. Les signaux transportés peuvent être soit directement d'origine numérique comme dans les réseaux de données, soit d'origine analogique (parole, image...) mais convertis sous une forme numérique. La tâche du système de transmission est d'acheminer le signal de la source vers le destinataire avec le plus de fiabilité possible, mais en réalité, ces lignes sont en général loin d'être parfaites. Pour cela, l'information devra être codée d'une manière spéciale permettant de déceler les erreurs, ou ce qui est encore mieux de les corriger automatiquement. L'enjeu du codage canal est de pouvoir détecter puis de corriger ces erreurs. Pour ce faire, Shannon a établi la théorie mathématique des communications. Il développe, dans son ouvrage, les règles du codage canal, et prouve l'existence d'une limite théorique qu'un code correcteur d'erreur puisse atteindre [1].

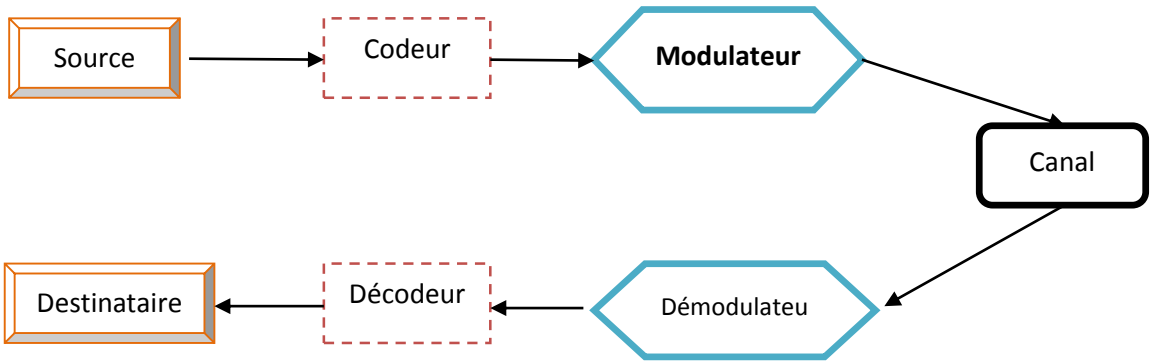


Figure 1. 1 : Schéma simplifié d'un système de transmission numérique

1.2. Notion de message numérique

Un message numérique est défini comme une suite d'éléments pouvant prendre une parmi **M** valeurs possibles. On appelle « alphabet » l'ensemble de ces valeurs. Les éléments, qui peuvent être aussi considérés comme des variables aléatoires discrètes, sont dits **M-aires**. Dans le cas particulier où l'alphabet est constitué uniquement de deux valeurs, notées traditionnellement **0** et **1**, les éléments sont dits binaires. Tout élément d'un message **M-aires** peut être représenté par une suite d'éléments binaires, ce qui justifie l'importance du cas binaire [1].

Un texte est un exemple de message numérique, composé d'éléments appartenant à un alphabet. Le message porteur de l'information émis par la source n'est pas nécessairement numérique par nature, ceci est même rarement le cas.

1.3. Théorie de l'information – Théorème de Shannon

On définit un canal de transmission comme un système physique permettant la transmission d'une information entre deux points distants. Le taux d'erreurs binaire (TEB) d'un message est le rapport du nombre de bits erronés par le nombre de bits du message.

$$\text{TEB} = \frac{\text{nombre de bits erronés}}{\text{nombre de bits du msg}} \tag{1.1}$$

En 1948, Shannon énonce dans « A Mathematical Theory of Information » le théorème fondamental de la théorie de l'information :

Tout canal de transmission admet un paramètre C, appelé capacité du canal, tel que pour tout $\varepsilon > 0$ et pour tout $R < C$, il existe un code de taux R permettant la transmission du message avec un taux d'erreurs binaire de ε.

En d'autres termes, on peut obtenir des transmissions aussi fiables que l'on veut en utilisant des codes de taux plus petits que la capacité du canal. Cependant, ce théorème n'indique pas le moyen de construire de tels codes.

On cherche donc à construire des codes ayant un taux le plus élevé possible (pour des raisons de temps et de coût) et permettant une fiabilité arbitrairement grande.

Les codes classiques présentés plus haut ne permettent pas d'atteindre cette limite [2].

1.4. Chaîne de transmission numérique

On définit un système de transmission numérique comme l'ensemble des fonctionnalités appliquées au transfert de messages entre une source et un récepteur [1]. Le schéma de principe d'une chaîne de transmission numérique est représenté sur la figure 1.2. On peut distinguer : la source de message, le milieu de transmission et le destinataire qui sont des données du problème. Le codage et le décodage de source, le codage et le décodage canal, l'émetteur et le récepteur représentent les degrés de liberté du concepteur pour réaliser le système de transmission. Nous allons maintenant décrire de façon succincte les différents éléments qui constituent une chaîne de transmission, en partant de la source de message vers le destinataire.

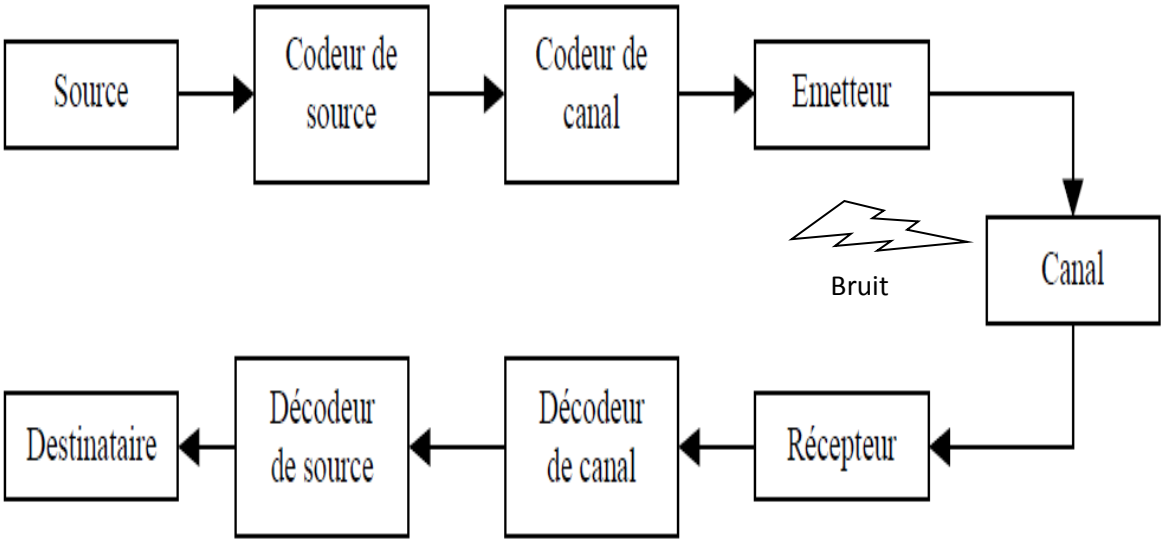


Figure 1. 2: chaine d transmission numérique

Le message porteur d'information émis par la source peut être de nature analogique (signal de parole, etc.). Dans ce cas, si le signal est analogique, il doit être numérisé, ce qui s'effectue en deux grandes étapes : **échantillonnage** (qui doit respecter le théorème d'échantillonnage pour garantir une représentation fidèle du signal), puis **quantification** de ces échantillons et représentation binaire des valeurs quantifiées.

1.4.1. Codeur de source

Le codage de source vise à la concision maximale du message afin de minimiser les ressources nécessaires à la transmission (temps, puissance, bande passante, surface de stockage, etc.). Ce codage peut donc, pour diminuer le coût de la transmission, substituer un message aussi court que possible au message émis par la source, dans la

mesure où cette substitution est réversible (i.e. que le message initial peut être exactement restitué) [1]. Le codage de source ne sera pas étudié. Indiquons simplement que ses limites théoriques sont fixées par la théorie de l'information (Premier théorème de Shannon).

1.4.2. **Codage canal**

Le codage canal vise, quant à lui, la protection du message contre les perturbations du canal de transmission.

Si les perturbations engendrées induisent une qualité de restitution (souvent mesurée quantitativement par la probabilité d'erreur par bit sortant du codeur de source ou par message, trame, etc.) incompatible avec les spécifications fixées, le codage canal se propose de transformer le message de manière à en augmenter la fiabilité de la transmission. Le « prix » qu'il en coûte est alors un accroissement de la taille du message.

Il y a donc antagonisme entre codage de source et codage canal, l'objectif du premier étant de diminuer la redondance du message de source, celui du deuxième d'en ajouter dans un but de protection. Nous verrons que la théorie de l'information fixe les conditions pour lesquelles cet antagonisme n'est qu'apparent, et la division des tâches entre codeur de source et codeur canal, pour l'instant arbitraire, justifiée [1].

a. **Objet et principe du codage**

La probabilité d'erreur dans un système de transmission est une fonction du rapport $\boldsymbol{S/B}$ de la liaison. Pour augmenter la qualité de la transmission, on est donc amené à augmenter ce rapport, soit en **augmentant la puissance du signal émis**, soit **en réduisant le facteur de bruit du récepteur**. Malheureusement, cela n'est pas toujours possible et on se heurte très vite à une limitation souvent d'ordre technologique ou économique.

Une alternative plus attrayante pour améliorer la qualité de la transmission est de faire usage du codage canal que l'on appelle communément **codage correcteur d'erreur**, bien que cette terminologie ne soit pas correcte au sens strict du terme. En effet, cette terminologie sous-entend qu'on laisse les erreurs **arriver** et qu'on les **corrige** ensuite. Or, cette façon de faire n'est pas inhérente au codage lui-même mais plutôt à la technique de décodage utilisée.

Bien entendu, il est encore plus intéressant d'empêcher les erreurs que de les corriger. Pour l'instant, nous allons donner le principe de base du codage.

Le codage canal consiste à introduire de la **redondance** dans le signal émis. La nécessité d'introduire de la redondance **pour protéger le signal émis** contre les erreurs de transmission est évidente. Supposons que l'on émette un message d'information binaire et que la détection se fasse bit par bit dans le récepteur. Si le message émis ne contient pas de redondance, chaque bit émis est essentiel et toute erreur de transmission conduit à une perte d'information irréversible. Si, au contraire, des bits de redondance sont introduits dans le message, on peut être en mesure de **détecter**, et même de **corriger**, des erreurs de transmission. Pour cela, **le décodeur teste si la loi de codage utilisée à l'émission est satisfaite au niveau de la séquence en sortie du circuit de décision**. Dans le cas où la loi est vérifiée, on décide qu'il n'y a pas eu d'erreur et on adopte définitivement les décisions du récepteur. Par contre, dans le cas où des erreurs sont détectées, le décodeur essaie de les localiser et ensuite de les corriger.

Une conséquence directe de la redondance inhérente au codage est l'accroissement du débit numérique et de la bande de fréquence occupée, à moins que la modulation soit modifiée en conséquence.

Les codes correcteurs d'erreurs peuvent être classés en deux grandes catégories : les codes en blocs et les codes convolutifs [2]. Dans notre travail, nous nous intéressons aux codes en blocs.

b. **Correction du canal**

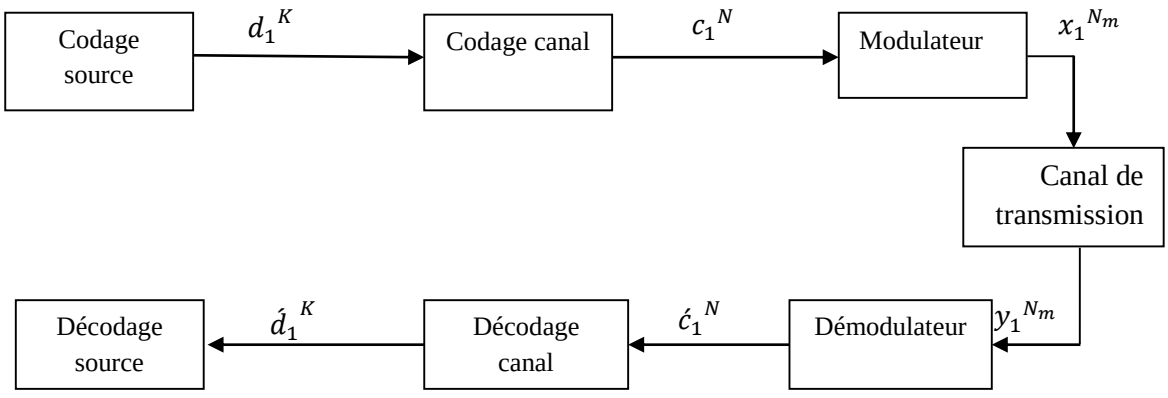


Figure 1. 3: modélisation d'une chaine de transmission numérique [3].

La correction canal implique d'ajouter des bits supplémentaires (**appelés bits de parité ou données de redondance**) par rapport aux bits d'information (alors appelés bits

systématiques) du codage source. Le message d'information d_1^k de K bits est transformé en un message codé c_1^Nde N bits. Le codage canal consiste donc à ajouter $M = N - K$ bits de redondance. On définit le rendement de codage R par la relation (1.2) [3].

$$R = \frac{K}{N} \tag{1.2}$$

Le codage canal pour un code C de longueur finie se caractérise par une fonction mathématique c. Lorsque cette fonction est une application linéaire, les codes correcteurs associés sont dits **linéaires**. Pour ce type de code, il existe une matrice G qui permet de définir la fonction de codage c. Cette matrice est nommée **Matrice Génératrice du code C**. La fonction de codage est définie par la relation (1.3).

$$c \rightarrow c = u.G \tag{1.3}$$

La distance de Hamming $d_H$est définie dans cet espace par la relation (1.4).

$$\forall\ c_{a_1}^N, c_{b_1}^N \in C$$

$$d_H(c_{a_1}^N 1, c_{b_1}^N) = \sum_{n=1}^{N} |c_{an} - c_{bn}| \tag{1.4}$$

La distance de Hamming permet d'établir une relation de distance entre deux éléments de C. Ainsi, la distance minimale (1.4) d'un code représente la plus petite distance de Hamming entre les mots d'un code. La linéarité des codes permet d'établir que la différence entre deux mots de codes est un mot de code. De ce fait, la distance de Hamming se résume à la seconde expression de l'équation (1.6).

$$\begin{aligned} d_{min} &= \min_{c_{a_1}^N 1, c_{b_1}^N 1 \in C} \sum_{n=1}^{N} |c_{an} - c_{bn}| \\ &= min_{c_1^N \in C} \sum_{n=1}^{N} |c_n| \end{aligned} \tag{1.5}$$

Sachant que les codes en blocs et les codes convolutifs sont des codes **linéaires,** ces premiers paramètres permettent de caractériser la plupart des codes correcteurs à travers le triplet(N, K, d_{min}).

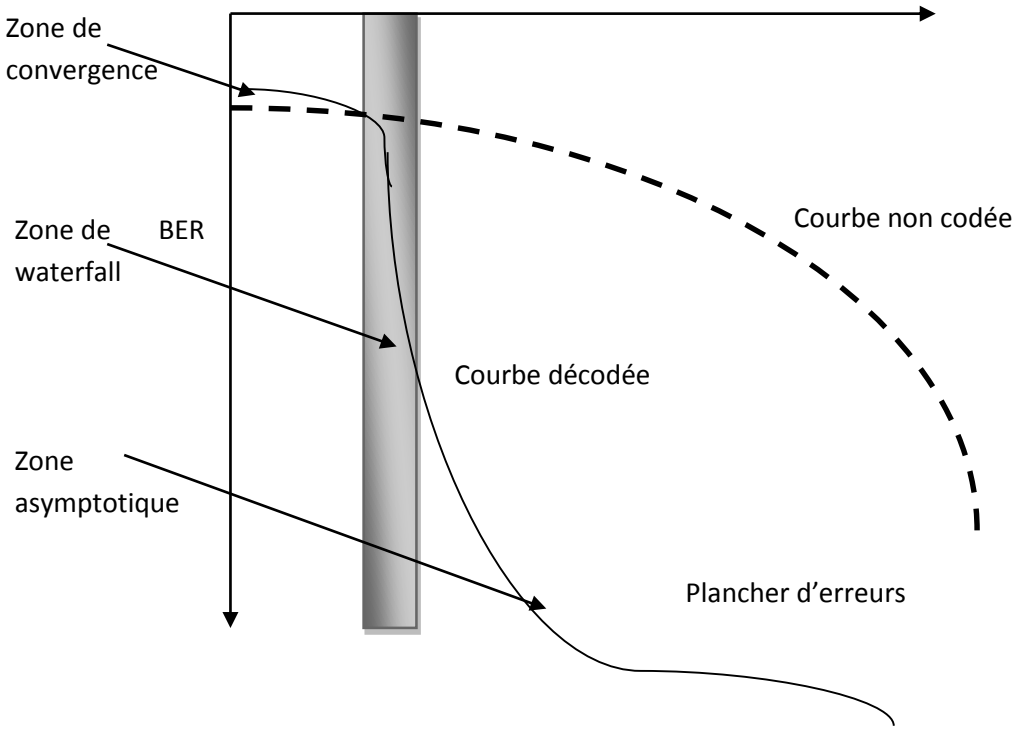


Figure 1. 4: Courbe de performance de décodage caractéristique.

Le codage canal est une opération qui permet de **réduire les erreurs** de propagation du canal. Il reste encore à établir les métriques permettant de quantifier ces erreurs de propagation. Le rapport signal à bruit (**SNR**) définit la différence entre l**a puissance du signal émis et le bruit du canal de propagation**. Celle-ci est exprimée en décibel (dB). La puissance du signal découle de l'énergie nécessaire à l'émission d'un symbole de modulation E_s La puissance de bruit dépend de la densité du spectre de bruitN_0. Ainsi, le SNR se définit suivant la relation (1.6).

$$SNR = \frac{E_s}{N_0}$$
$$SNR_{db} = -20log_{10}\left(\frac{E_s}{N_0}\right) \quad (1.6)$$

Afin de comparer les performances en sortie de décodage, on s'intéresse non pas au rapport SNR, mais au rapport E_b/N_0 qui rend compte de l'énergie E_b nécessaire à l'émission d'un élément binaire de la source d_K par rapport au bruit du canal. Dans ce cas, l'énergie binaire vérifie la relation (1.7) [3].

$$E_b = \frac{E_s \times N}{n_m \times K} \quad (1.7)$$

1.4.3. **Modulation**

Le message numérique issu du codeur canal, en tant que suite d'éléments binaires, est une grandeur abstraite.

Pour transmettre ce message, il est donc nécessaire de lui associer une représentation physique sous forme d'un signal électrique. C'est la première fonction de l'émetteur, appelée généralement opération de modulation.

la modulation consiste à associer à chaque mot de n éléments binaires issu du message, un signal $s_i(t)$, i =1…, $M = 2^n$ de durée $T = nT_b$.

Le choix du type de signaux dépend des propriétés physiques du milieu de transmission que le signal va traverser. L'émetteur assure donc aussi une fonction d'adaptation du signal modulé au milieu de transmission.

Parmi les autres traitements effectués par l'émetteur, on peut citer le filtrage du signal modulé pour limiter sa bande. Lorsque la bande allouée à la transmission est centrée autour d'une fréquence élevée, le modulateur élabore parfois un signal dont le spectre est centré autour d'une fréquence dite intermédiaire plus basse, et l'émetteur effectue ensuite un changement de fréquence qui permet de centrer le signal modulé autour de la fréquence souhaitée [1].

L'étape de conversion numérique/analogique intervient au cours des traitements pris en charge par l'émetteur, la tendance actuelle étant de « retarder » de plus en plus cette étape. De plus en plus de systèmes effectuent la modulation en numérique jusqu'à la fréquence intermédiaire. Les dernières étapes du traitement de l'émetteur sont un dernier filtrage d'émission (analogique à la fréquence finale) et l'amplification. Où nous développerons un modèle équivalent.

a. **Différents type de modulation**

Cette section s'intéresse aux modulations numériques les plus courantes. La modulation numérique correspond à l'adaptation des signaux du domaine numérique au domaine analogique. Ce phénomène se traduit par la modulation d'une onde sinusoïdale en amplitude ou en phase. Cette transformation est représentée par une information complexe. Un diagramme de constellation aussi nommé diagramme IQ représente, dans ce cas, le codage des éléments binaires en éléments complexes. On distingue plusieurs types de modulations. Les modulations par saut de **phase (PSK)** représentent les nm éléments de modulation sur un cercle de même amplitude et répartis suivant les racines nièmes de l'unité. On distingue en particular la modulation BPSK (Binary Phase-Shift

Keying) , QPSK (Quadratic Phase-Shift Keying), 8-PSK, ... Pour ces types de modulation, l'amplitude de chaque élément est constante et donnée en fonction de l'énergie nécessaire à l'émission d'un symbole numérique. La figure 1.5 représente différents diagrammes de constellation associés à ces modulations. La propagation du canal entraîne une déformation de la constellation entre l'émission et la réception [1].

Le démodulateur établit une première décision à la réception en fonction du diagramme IQ. Pour ce faire, le diagramme de constellation est séparé en zones de décisions.

La Figure (1.5) montre les secteurs de décision suivant ce principe pour les modulations BPSK et QPSK. Cette décision au niveau binaire est appelée décision dure. La décision dure entraîne une forte perte d'information des éléments reçus. En effet, la notion de distance entre un élément reçu $y_n$et sa décision dure est perdue. On peut donc améliorer cette décision en définissant une relation de distance entre les éléments [1].

On définit le Log-Rapport de Vraisemblance noté Lc (ck) comme le rapport de probabilités entre les différentes décisions sur les éléments binaires

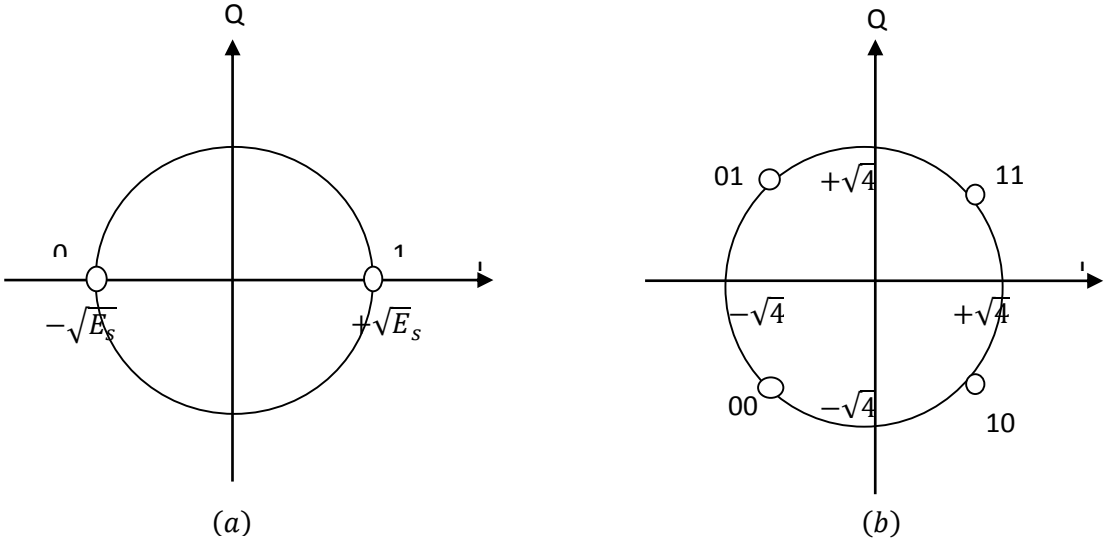


Figure 1. 5: Diagramme de constellation pour une modulation BPSK (a) et QPSK (b) en fonction des informations du canal.

b. **Modulation par impulsion codé (MIC)**

La modulation MIC permet de transmettre les informations analogique (paroles, son..) sur une voie numérique qui consiste à échantillonner un signal, quantifier chacun des échantillons et les transmettre.

Les caractéristiques de la technique MIC :

- 256 niveaux de quantification ⟶ 8 bits pour chaque échantillon
- Fréquence d'échantillonnage 8000hz ⟶ un échantillon chaque 125 µs.
- Ce qui autorise un débit de transmission égal à 64 kbps

1.4.1. Canal de transmission

Le canal, au sens des communications numériques, est comme représenté à la Figure 2. Par la suite, aussi bien dans la présentation des résultats de la théorie de l'information que dans cette introduction au codage canal, nous utiliserons un modèle canal plus global, incluant une partie de l'émetteur et du récepteur.

Le canal inclut le milieu de transmission (lien physique entre l'émetteur et le récepteur : câble, fibre, espace libre), et les interférences (provenant des autres utilisateurs du milieu de transmission, de brouilleurs intentionnels ou non), et représente les différentes perturbations engendrées lors de la transmission du signal entre une source et le récepteur (perturbation aléatoire issue du milieu des équipements électroniques) [1]. Cette représentation caractérise toutes les déformations physiques de la transmission. Il existe de nombreuses représentations des perturbations du canal [1].

1.4.2. Démodulateur

Le démodulateur réalise la transformation du signal reçu du support et un signal sous forme « rectangulaire ».il s'aide notamment d'un échantillonneur et une horloge qui permettent de déterminer les intervalles significatifs.

1.5. Le décodeur canal

Plusieurs stratégies différentes peuvent être utilisées par le décodeur canal.

La première est la détection d'erreurs. Le décodeur observe la séquence reçue (ferme ou souple) et détecte la présence éventuelle d'erreur. Cette détection peut servir à contrôler le taux d'erreur (Error Monitoring) ou à mettre en œuvre des techniques de retransmission

(ARQ : Automatic Repeat Request) : le décodeur demande à l'émetteur de retransmettre la séquence dans laquelle une erreur a été détectée. Il est évident que ce type de procédé nécessite une voie de retour. Cette stratégie de détection est surtout utilisée par les couches transport et supérieures du modèle OSI. Les techniques employées ne seront que mentionnées, dans la mesure où il ne s'agit que d'un cas particulier et simple de la seconde stratégie. La deuxième est la correction d'erreurs (FEC : Forward Error Correction). Elle nécessite des algorithmes beaucoup plus complexes que la simple détection, et plus de redondance dans la séquence émise. Toutefois, le milieu de transmission est utilisé de manière plus efficace. Cette stratégie sera celle principalement étudiée dans notre projet. Elle est omniprésente au niveau couche physique de tous les systèmes de transmission et d'enregistrement [1].

1.6. Conclusion

La théorie de l'information fournit un modèle mathématique permettant de quantifier l'information émise par la source d'une communication. Le codage de source a pour but de transmettre le contenu informatif du message en économisant les bits. La correction canal est devenue une étape essentielle pour protéger les données émises sur les canaux de transmission bruités. Les stratégies de codage s'adaptent cependant en fonction des contextes de transmission et du type de données transmises. Dans ce chapitre, un état de l'art montre la multiplicité des stratégies de codage canal dans les différents standards de transmission et établit des pistes de convergence pour lesquelles des récepteurs doivent être considérés. Les informations traitées au niveau du codage (ou de l'encodage) sont sous forme de symboles binaires (0 ou 1) ou de blocs de symboles binaires formant ainsi des mots. Le codage canal rajoute au message des bits qui seront utilisés par un algorithme de réception pour déterminer s'il y a des erreurs et les corriger éventuellement. Le codage canal ajoute de la redondance, alors que le codage de source a pour tâche de l'éliminer.

Chapitre 2 **Codes de détection et de correction d'erreurs**

2.1. Introduction

Un système de transmission numérique est conçu pour modéliser le transfert d'informations numériques sur des canaux de transmissions bruités. Concrètement, lorsqu'une information est transmise sur un canal, elle est soumise à de nombreuses perturbations physiques qui entraînent une déformation à la réception du signal transmis. Des erreurs résultent de ces perturbations et le message reçu n'est pas considéré comme fiable. Pour cela, le principe du codage correcteur d'erreurs (ou codage canal) est d'ajouter aux données utiles, des données redondantes de manière à rendre plus fiable la transmission des données utiles. Parmi les types du codage les plus utilisées, nous citons : les codes convolutifs, les turbo codes ainsi que les codes en bloc.

2.2. la famille des codes correcteurs d'erreurs

Dans le domaine de la théorie de l'information, les codes sont utilisés pour détecter et corriger les erreurs de transmission. Pour cela, il existe deux grandes familles de codes correcteurs : code en treillis et code en bloc, comme le montre la figure suivante[4].

Dans notre projet, nous nous intéressons au codage de Hamming.

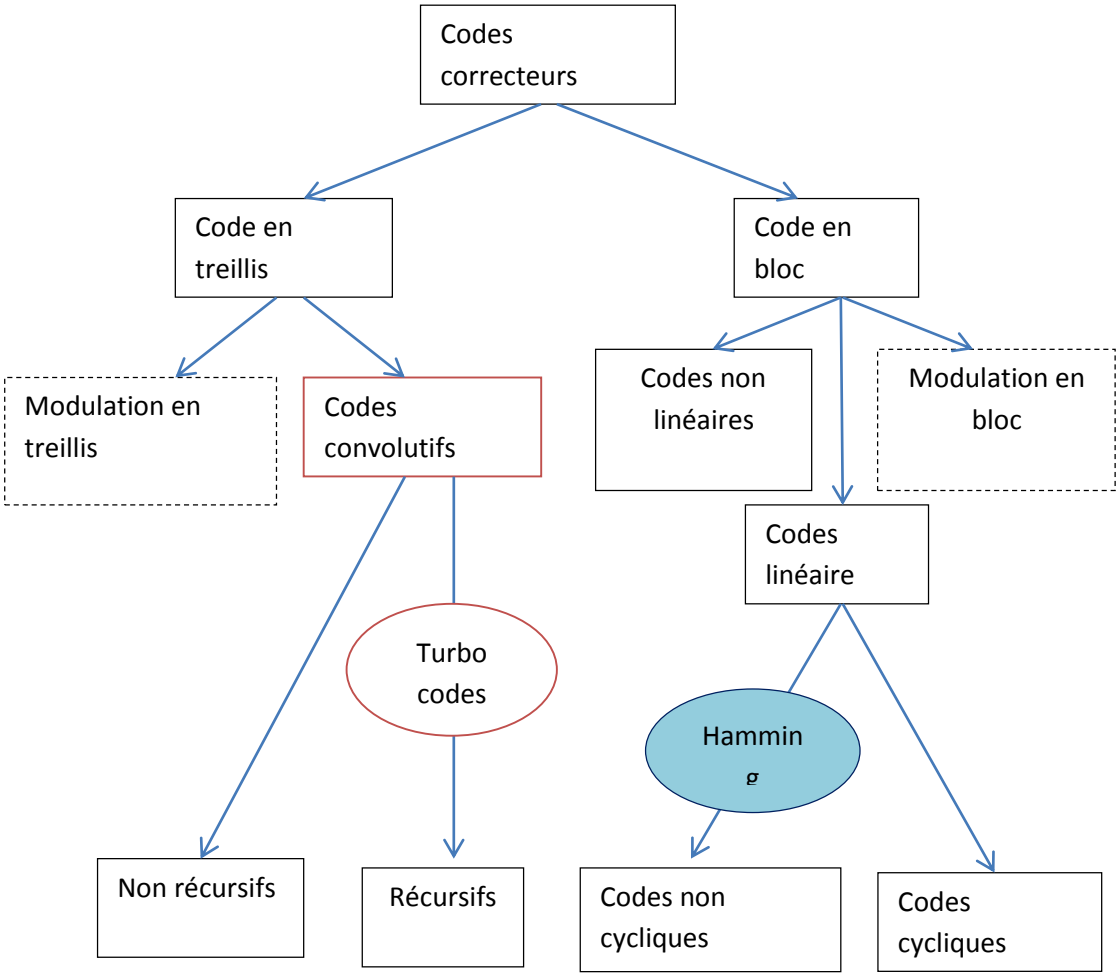


Figure 2. 1: l'arbre des codes correcteurs.

2.2.1. **Code en bloc**

Le codage en blocs consiste à associer, à chaque bloc de $\boldsymbol{k}$ bits d'information, un bloc de n bits ($\boldsymbol{n} > k$) contenant $\boldsymbol{n} - \boldsymbol{k}$ bits de redondance. Les 2^k blocs de $\boldsymbol{n}$ bits délivrés par le codeur sont appelés les mots de code. Le rapport $\boldsymbol{k}/\boldsymbol{n}$ est appelé le rendement de codage. Par la suite, nous utiliserons la notation $\boldsymbol{v}(\boldsymbol{n}, \boldsymbol{k})$ pour désigner un code ayant une longueur de bloc $\boldsymbol{n}$ et $\boldsymbol{k}$ bits d'information par bloc. Les opérations de codage et de décodage dans les codes en blocs se font à l'aide d'additions et de multiplications sur des éléments binaires. Ces dernières correspondent respectivement aux opérations logiques ET et OU exclusif [7].

Définition 1 : Le rendement (binaire) d'un code en bloc de longueur n, aussi appelé taux de codage (équation 2.1).

$$R = \frac{\log_2(|C|)}{n} \tag{2.1}$$

où $|C|$ désigne le nombre de mots du code C.

Nous remarquons que log_2 $(|C|)$ est le nombre de bits à utiliser pour étiqueter tous les mots de C.

La distance minimale d'un code en bloc est un paramètre très important qui mesure, entre autres, si un code est « bon » ou pas. Avant de donner la définition de la distance minimale, nous introduisons la distance de Hamming. Elle mesure le nombre de symboles par lesquels deux mots diffèrent. On définit tout d'abord la distance de Hamming entre deux symboles comme étant égale à 0 si les deux symboles sont identiques et égale à 1 sinon. Soient alors les mots $x = (x_1, x_2, \ldots, x_n)$ et $y = (y_1, y_2, \ldots, y_n)$. La distance de Hamming entre ces deux mots est donnée par l'équation suivante :

$$d_H(x, y) = \sum_{i=1}^{n} d_H(x_i, y_i) \tag{2.2}$$

avec $d_H(x, y)$ la distance de Hamming entre les symboles x_i et y_i. La distance minimale d'un code en bloc C est la plus petite distance de Hamming entre deux mots distincts de C.

Définition 2 : La distance minimale d_{min}du code en bloc C est est donnée par l'équation 2.3[7].

$$d_{min}(c) = min_{x,y \in c, \frac{x}{y}} d_H(x, y) \tag{2.3}$$

Quelques exemples de codes en blocs **:** nous allons donner quelques exemples de codes en blocs en commençant par les plus connus.

a. **Code de parité**

Ce code utilise un seul bit de redondance par mot de code, déterminé de façon à avoir :

$$\sum_{i=0}^{i=n-1} m_i \tag{2.4}$$

où les $\boldsymbol{m_i}$, $i = 0, 1, .., n - 1$, désignent les n bits du mot de code. Autrement dit, la somme des n bits d'un mot de code dans un code de parité est paire. La distance

minimale de ce code est de 2, ce qui ne permet pas de corriger d'erreurs. Il permet, par contre, de détecter une erreur isolée (ou un nombre impair d'erreurs) par bloc. La matrice génératrice $\boldsymbol{G}$ de ce code est :

$$G = \begin{bmatrix} 1\,0\,0\ \ldots\ldots\ldots\ 0\,1 \\ 0\,1\,0\ \ldots\ldots\ldots\ 0\,1 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 0\ \ldots\ldots\ldots\ldots\ 0\,1\,1 \end{bmatrix} \tag{2.5}$$

La matrice de contrôle de parité $\boldsymbol{H}$ se réduit à un vecteur : $\boldsymbol{H} = [1\ 1\ldots\ldots\ldots\ldots 1]$

Le calcul du syndrome consiste simplement à calculer la somme des n éléments du mot reçu et à tester sa parité. Si cette somme est paire, on déclare qu'il n'y a pas d'erreurs [8].

b. **Code à répétition**

Il s'agit d'un code très simple dans lequel on répète n fois chaque bit du message à envoyer.

On a alors $\boldsymbol{C} = \{(\mathbf{00}\ldots\mathbf{0});\ (\mathbf{11}\ldots\mathbf{1})\}$.

Illustration : Considérons le cas où $\boldsymbol{n} = \mathbf{5}$, on obtiendrait alors le mot de code :

$$\mathbf{11111\ 00000\ 11111\ 11111\ 00000\ 00000\ 11111}$$

La distance minimale est précisément la longueur du code n, ce qui donne la capacité de détecter $n-1$ erreurs et de corriger $[(n-1)/2]$ erreurs. Ce code a pour matrice génératrice :

$$\text{G= [1 1 .....1]} \tag{2.6}$$

La matrice de contrôle de parité :

$$\text{H} = \begin{bmatrix} 1\,1\,0\,0\ \ldots\ldots\ldots\ldots 0 \\ 1\,0\,1\,0\ldots\ldots\ldots 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 1\,0\,0\,0\ldots\ldots 0\,1 \end{bmatrix} \tag{2.7}$$

Pour ce code également, le décodage est très simple. Si le nombre de 1 dans le mot détecté est supérieur au nombre de 0, on déclare que le bit d'information émis est 0. On déclare un 1 dans le cas contraire [8].

2.2.2. **Code de Hamming**

Un code de Hamming permet la détection et la correction automatique d'une erreur si elle ne porte que sur une lettre du message. Un code de Hamming est parfait, ce qui signifie que, pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est maximal.
Pour les codes de Hamming, la longueur n et le nombre de bits d'information k sont de la forme $n = 2^{\mathrm{m}} - 1$ et $k = 2^{\mathrm{m}} - \mathrm{m} - 1$ pour m entier. Les colonnes de la matrice de contrôle de parité sont les représentations binaires des nombres de 1 à n. Chaque colonne est constituée de $m = n - k$ éléments. La distance minimale d'un code de Hamming est de 3, quelle que soit la valeur des paramètres n et k. Le code permet donc de corriger une erreur et d'en détecter deux. Le décodage est très simple car, en présence d'une erreur, le syndrome prend la valeur de la colonne de la matrice de contrôle parité correspondant au rang de l'erreur. Connaissant les colonnes de la matrice de contrôle parité, on en déduit la position de l'erreur que l'on peut alors corriger.

Exemple :
Considérons le code de Hamming correspondant à $n = 7$ et $k = 4$. La matrice de contrôle parité est :

$$H = \begin{bmatrix} 1\,1\,1\,0\,1\,0\,0 \\ 1\,1\,0\,0\,0\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1 \end{bmatrix} \tag{2.8}$$

Et la matrice génératrice $\boldsymbol{G}$:

$$G = \begin{bmatrix} 1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1\,0\,1 \\ 0\,0\,0\,1\,0\,1\,1 \end{bmatrix} \tag{2.9}$$

Supposons que le mot de code émis soit $\boldsymbol{c} = [0\,0\,0\,1\,0\,1\,1]$et que le mot détecté soit
$\boldsymbol{r} = [1\,0\,0\,1\,0\,1\,1]$ Comportant ainsi une erreur à la première position.
Le syndrome **S** est alors égal à la première colonne de $\boldsymbol{H}$, c'est-à-dire, nous avons $\boldsymbol{S} = [1\,1\,1]$ [8].

a. **Distance de Hamming**

La distance de Hamming $d(\boldsymbol{x}, \boldsymbol{y})$ entre une telle paire de vecteurs est définie comme le nombre d'emplacements dans lesquels leurs éléments respectifs diffèrent. Le poids de Hamming $\boldsymbol{w}(\boldsymbol{x})$ d'un vecteur du code x est défini comme le nombre d'éléments non zero dans le vecteur du code. La distance minimale $\mathbf{d_{min}}$ d'un code du bloc linéaire est définie comme la plus petite distance de Hamming entre toute paire de vecteurs du code.

Comme la somme (ou différence) de deux vecteurs de codes est un vecteur du code, nous pouvons affirmer que la distance minimale d'un code du bloc linéaire est le plus petit poids de Hamming du vecteur du code non nul dans le code

La validité d'un code est définie par sa complexité et par sa capacité de correction. Un code est d'autant plus optimal qu'il permet de corriger un nombre important d'erreurs au prix d'une complexité réduite. Cette capacité de correction peut être calculée à partir de la distance de Hamming minimale séparant deux mots distincts du code.

b. **Pouvoir de détection et de correction**

- **Détection d'erreur**

Les codes correcteurs d'erreur sont aussi souvent utilisés pour détecter des erreurs. Lorsqu'ils sont utilisés en détection d'erreur, le décodage se limite alors à répondre à une seule question : le mot reçu est-il un mot de code ? [9]

Définition :

On dit qu'un code en bloc détecte t′ erreurs si et seulement s'il peut détecter toutes les configurations d'erreurs dont le nombre est inférieur ou égal à t'. On en arrive rapidement au résultat suivant :

Résultat :

(Capacité de détection d'un code) Un code en bloc de distance minimale d_{min} est capable de détecter t′erreurs si t′ vérifie

$$\mathbf{t' = d_{min} - 1 \text{ (pouvoir de détection)}} \qquad (2.10)$$

En effet, le décodeur détecte qu'il y a une ou plusieurs erreurs si le mot reçu n'appartient pas au code. Or, le codeur transmet un mot de code. Il y a non détection des erreurs lorsque le mot reçu est un autre mot de code. Un autre mot de code étant à une

distance de Hamming au moins égale à d_{min} du mot émis. Pour pouvoir toujours détecter les erreurs, il suffit que leur nombre reste strictement inférieur à d_{min}[9].

- **Correction d'erreur**

Nous nous intéressons ici au nombre d'erreurs qu'un code peut corriger. Tout d'abord, nous donnons une définition de la capacité de correction d'erreur d'un code.

Définition : On dit qu'un code en bloc C corrige t erreurs s'il peut corriger toutes les configurations d'erreurs dont le nombre est inférieur ou égal à t.

Supposons maintenant que le canal introduit t erreurs. Ceci signifie que, si x est le mot émis et y le mot reçu, alors $\mathbf{d_H(y,x) = t}$ **,** le décodeur corrige. Le mot décodé sera le mot émis x. Donc, pour pouvoir décoder x, il faut qu'il n'y ait aucun mot x′ dans le code tel que $d_H(y,x') \leq t$. Supposons qu'un tel mot x′ existe. La distance de Hamming entre x et x′ peut être bornée en utilisant l'inégalité triangulaire :

$$d_H(x,x') \leq d_H(y,x) + d_H(y,x') \leq 2t. \quad (2.11)$$

Supposons maintenant que $d_{min}(c) \geq 2t + 1$. Alors, la condition (9) ne peut plus être vérifiée et un tel x′ n'existe pas, ce qui fait que les t erreurs seront toujours corrigées. Nous en arrivons au résultat suivant :

Résultat : (Capacité de correction d'un code) Un code de distance minimale d_{min} peut corriger t erreurs si t vérifie

$$t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor \quad \text{(pouvoir de correction)} \quad (2.12)$$

où $\lfloor u \rfloor$est la partie entière de u.

Ainsi, pour corriger une erreur simple, il faut une distance minimale au moins égale à 3. Pour 2 erreurs, il faut une distance au moins égale à 5, etc [9].

2.2.3. **Matrice génératrice**

Considérons un code bloc linéaire q-aire $c\ (n,k)$. On peut représenter ce code comme un sous- espace vectoriel de dimension k d'un espace vectoriel q-aireV_q^nde dimension n.

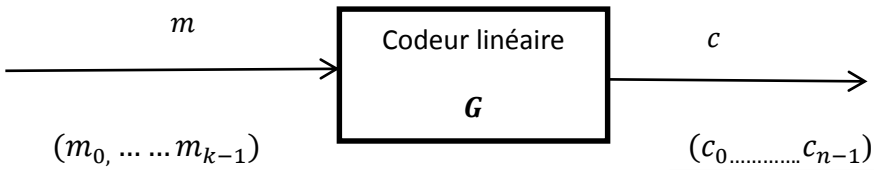


Figure 2. 2 : codeur linéaire.

Le processus de codage consiste à assigner à chacun des q^k (vecteurs) messages q-aires possibles, m: $\{(m_{0,}\mathrm{m}_1 \dots \dots m_{k-1})\}$, un (vecteur) mot code q-aire du code : $\{c_0, c_{1\dots\dots\dots}c_{k-1})\}$. Les messages sont donc des vecteurs de longueur K alors que les mot-code correspondants sont des vecteurs de longueurs n.

Soit $\{g_0, g_1, \dots, g_{(k-1)}\}$ un ensemble de K vecteurs formant une base de mot code de longueur n [10]:

$$\begin{aligned} g_0 &= [\, g_{0,0}\, g_{0,1}, \dots \dots g_{0,n-1}] \\ g_0 &= [\, g_{1,0}\, g_{1,1}\, , \dots \dots , g_{1,n-1}] \\ &\vdots \\ g_{k-1} &= [g_{k-1,0}\, , g_{k-1,1}, \dots \dots , g_{k-1,n-1}] \end{aligned} \tag{2.13}$$

On peut construire une matrice G à l'aide des K vecteurs de base $\{g_0, g_1, \dots, g_{(k-1)}\}$:

$$G=\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix}=\begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{2.14}$$

Cette matrice G, formée par le vecteur de la base, est appelée matrice génératrice de code C. La matrice génératrice peut être utilisée pour générer les q^k mot- code des q^k messages m: $\{(m_{0,}\mathrm{m}_1 \dots \dots m_{k-1})\}$:

$$C = mG \tag{2.15}$$

La matrice $\boldsymbol{G}$ à k lignes et à n colonnes est appelée la matrice génératrice du code $\boldsymbol{c}(n, k)$. La matrice génératrice d'un code en blocs linéaire n'est pas unique. En permutant des lignes et des colonnes, ou encore en faisant un changement de base dans l'espace, il est toujours possible d'écrire la matrice $\boldsymbol{G}$ sous la forme :

$$\mathrm{G}=[P.I_k.]=\begin{bmatrix} p_{00} & p_{01} & \cdots & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ p_{k-1,0} & \dots & & & p_{k-1,n-k-1} \end{bmatrix} \left| \begin{matrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{matrix} \right| \tag{2.16}$$

Où $\boldsymbol{I_k}$ est la matrice identité de dimension $k \times k$ et $\boldsymbol{P}$ une matrice de dimension $k\ x\ (n - k)$ utilisée pour calculer les $n - k$ bits de redondance.

Ainsi écrite, la matrice génératrice $\boldsymbol{G}$ engendre des mots de code de la forme :

$$c = [m, mP] \tag{2.17}$$

Les k bits d'information et les $n - k$ bits de redondance étant ainsi séparés, le code correspondant est systématique [10].

2.2.4. **Matrice de contrôle de parité**

À chaque matrice génératrice $\boldsymbol{G}$ de dimension $k \times n$, on peut associer une matrice $\boldsymbol{H}$ de dimension $(n-k) \times n$ telle que les lignes de $\boldsymbol{G}$ soient orthogonales à celles de $\boldsymbol{H}$, c.à.d :

$$\mathrm{G}H^T = 0 \tag{2.18}$$

où l'exposant T désigne la matrice transposée. L'orthogonalité entre deux vecteurs $x = (x_0, x_1, \ldots, x_{n-1})$ et $y = (y_0, y_1, \ldots, y_{n-1})$ signifie ici que le produit scalaire est nul, c'est-à-dire :

$$\sum_{i=0}^{n-1} x_i y_i = 0 \tag{2.19}$$

En utilisant l'expression réduite de la matrice $\boldsymbol{G}$, donnée par (2.18), l'équation (2.20) conduit à choisir la matrice $\boldsymbol{H}$ de la forme :

$$\mathrm{H}[I_{n-k}, P^T] = \left[\begin{array}{cccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 0 & p_{00} & p_{01} & \cdots & \cdots & p_{k-1,0} \\ 0 & 1 & 0 & 0 & 0 & 0 & p_{01} & p_{11} & \cdots & \cdots & p_{k-1,1} \\ 0 & 0 & 1 & 0 & \cdots & 0 & \vdots & \cdots & \cdots & \cdots & p_{k-1,2} \\ \vdots & \vdots & \vdots & 1 & \cdots & \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & 1 & p_{0,n-k-1} & \cdots & \cdots & \cdots & p_{k-1,n-k-1} \end{array}\right] \tag{2.20}$$

Tout mot de code est orthogonal aux lignes de la matrice de contrôle parité $\boldsymbol{H}$ du code. Cette propriété découle directement des équations (2.16) et (2.20) :

$$vH^T = mGH^T = 0 \tag{2.21}$$

Elle permet de tester si une séquence donnée est un mot de code [8].

2.2.5. **Syndrome**

La correction des erreurs peut ainsi être réalisée à partir du syndrome **s** dont le nombre de configurations $2^{(n-k)}$ est généralement très inférieur aux 2^k mots de code. Le nombre de configurations non nulles du vecteur d'erreur $\boldsymbol{e}$, à savoir 2^{n-1}, étant

supérieur au nombre de configurations du syndrome, plusieurs configurations du vecteur $\boldsymbol{e}$ peuvent conduire à une même valeur du syndrome. La règle de décodage peut alors consister à choisir, pour chaque valeur non nulle du syndrome, la configuration d'erreur de poids minimal.

On peut montrer que cette façon de procéder est équivalente à la recherche systématique du mot le plus vraisemblable. En effet, pour un code en blocs pouvant corriger τ erreurs, toutes les configurations d'erreurs de poids inférieur ou égal à τ correspondent à des valeurs différentes du syndrome. Le mot du code le plus vraisemblable est alors égal à $\boldsymbol{r} - \boldsymbol{e}$ (modulo 2).

Notons finalement que l'utilisation du syndrome pour réaliser le décodage peut aussi s'avérer difficile à mettre en œuvre lorsque la différence $(n - k)$ excède quelques unités. Dans ce cas, on utilise généralement des algorithmes de décodage spécifiques à certaines classes de codes, que le lecteur peut trouver dans les ouvrages spécialisés.

a. **Décodage Par Syndrome**

Une autre méthode permettant le décodage des codes linéaires est la méthode de décodage par syndrome.

Considérons un code bloc linéaire $C\ (n, k)$ et sa matrice de contrôle H. Un vecteur **r** est un mot-code valide si et seulement si $\mathbf{r}\boldsymbol{H}^{\boldsymbol{T}} = \mathbf{0}$ (autrement dit, le code $C(\ n, k)$ constitue l'espace nul de la matrice H)

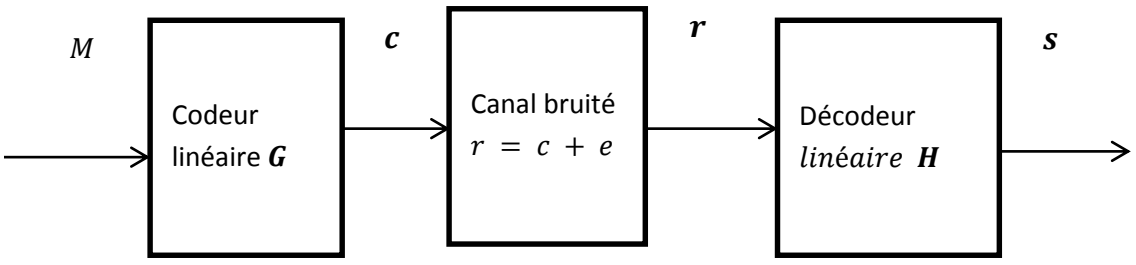


Figure 2. 3 : **chaine du codage du $\boldsymbol{M}$ vers $\boldsymbol{s}$**

Si le mot- code reçu **r** est un mot-code valide, alors, en le multipliant avec la matrice de control (transposée) $\boldsymbol{H}^{\boldsymbol{T}}$, on obtient $\mathbf{r}\boldsymbol{H}^{\boldsymbol{T}} = \mathbf{0}$

Comme nous l'avons vu, ceci se produit lorsque **r = c**, i.e. le vecteur reçu est le mot-code original, ou lorsque **r = $\grave{\boldsymbol{c}}$** , avec **$\grave{\boldsymbol{c}}$ =c,** et la séquence d'erreur ne peut être détectée.

La plupart du temps, cependant, les erreurs introduites par le canal résultent en un vecteur à la réception **r = c + e** ou e≠0 Et **e** ≠∀≠ $\boldsymbol{c}$. Alors :

$$r\boldsymbol{H^T} = (\boldsymbol{c} + \boldsymbol{e})\,\boldsymbol{H^T} = \boldsymbol{cH^T} + \boldsymbol{eH^T} = \mathbf{0} + \boldsymbol{eH^T} = \boldsymbol{eH^T} = \boldsymbol{s}$$

Où le vecteur s de langur **(n-k)** est le syndrome du vecteur reçu r et ne dépend que du vecteur d'erreur $\boldsymbol{e}$, qui se est présent à l'entrée du décodeur [10].

2.3. Principes du décodeur

Le rôle de n'importe quel décodeur est de retrouver, à partir de la sortie du canal, les bits qui ont été transmis. Le décodage se fait en deux étapes :

✓ A partir du mot reçu (qui correspond à celui à la sortie du canal), trouver un mot de code qui paraît être le plus probable (nous verrons que cela a un sens très précis).

✓ A partir du mot de code trouvé à l'étape 1, en déduire les bits d'information qui ont été transmis. Commençons par un exemple, celui du code à répétition de longueur 3 sur le canal binaire symétrique [11].

2.3.1. Le maximum de vraisemblance

Le critère de décodage le plus utilisé est le critère du maximum de vraisemblance, qui peut être évalué sur tous les modèles de canaux. C'est un critère très important qui peut être utilisé dans d'autres applications que le codage, en fait, dans tout problème de détection (détection d'un visage dans une image, détection de symboles, reconnaissance du locuteur, détection d'anomalie, détection de présence, etc). On suppose, que le mot de longueur n émis est le mot x. Le mot reçu, après passage dans le canal, est le mot y [9].

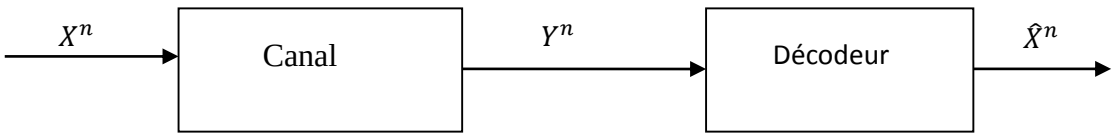


Figure 2. 4: Décodage

La variable aléatoire qui désigne l'entrée du canal est la variable X. Le vecteur aléatoire composé de n copies de la variable X sera noté X^n. De même, la variable aléatoire qui désigne l'entrée du canal est la variable Y et le vecteur aléatoire composé de n copies de la variable Y sera noté Y^n.

La figure 6 représente de façon schématique le décodeur avec toutes les variables aléatoires. On suppose que la règle de décodage vérifie les propriétés suivantes :

• L'ensemble de tous les vecteurs y pour lesquels le décodeur choisit le mot de code x sera noté Ω_x.

• La règle de décodage est une règle déterministe.

• La règle de décodage est celle qui minimise la probabilité d'erreur par mot après décodage (ce qui semble raisonnable) [9].

2.4. Conclusion

Pour garantir la qualité de la communication, il faut bien connaître le canal afin de déterminer la probabilité d'erreur et de ne rajouter que les bits strictement nécessaires

Un exemple simple de codage canal consiste à doubler l'information, c'est-à-dire à envoyer deux fois les données à émettre de sorte que, si une partie des données est mal détectée dans la première séquence, elle puisse être récupérée dans la deuxième. Pour les codes utilisés en pratique, chaque donnée redondante est calculée en fonction de chacune des données de l'information utile. Cela permet, en plus d'apporter de la redondance aux informations, de les sécuriser puisqu'il est nécessaire en réception de connaître la règle pour décoder l'information reçue. Les codes correcteurs d'erreur sont regroupés suivant leurs caractéristiques et leurs propriétés. La première grande caractéristique concerne leur propriété de linéarité.

Une autre propriété concerne le caractère systématique d'un code.

Chapitre 3 **Initiation Arduino**

3.1. Introduction

La carte Arduino est très populaire parmi les roboticiens, les débutants, les amateurs et les professionnels. Son succès est principalement dû à sa conception de type ouverte et à sa facilité extrême d'utilisation. De plus, le fait qu'il s'agisse d'une plate-forme abordable qui peut être programmée en utilisant une IDE facile à utiliser en Open Source lui a obtenu le soutien d'une communauté importante et très active.

Tout cela le rend très simple pour débuter avec l'Arduino grâce aux très nombreux projets et matériels d'apprentissage facilement accessibles.

3.2. Historique

L'Arduino est né en 2005 dans le cadre d'un projet étudiant en Italie. Massimo Banzi, architecte logiciel, est co-fondateur du projet ; son leitmotiv, la démocratisation de l'ingénierie. Pour de multiples raisons (manque de subventions, fermeture de **l'Interaction Design Institute**, ...), les sources aussi bien matérielles que logicielles furent libérées.

C'est ainsi qu'un modeste projet d'étudiant bouleversa le milieu de l'électronique, en rendant la technologie plus accessible. Aujourd'hui, vous pouvez acquérir une carte Arduino pour 25 Euros ou la construire vous-même sans avoir de droit de licence à payer. Tous les schémas et les codes sources sont diffusés sous licence libre.

L'Arduino est devenu presque un réflexe dès lors qu'on souhaite s'initier au monde de l'électronique numérique. Il a permis de démocratiser ce domaine et surtout d'en faciliter l'accès [12].

3.3. Les Cartes Arduino

Les cartes Arduino possèdent un microcontrôleur facilement programmable ainsi que de nombreuses entrées-sorties. Plusieurs cartes Arduino existent et qui se différencient par

la puissance du microcontrôleur ou par la taille et la consommation de la carte. Le choix du type de carte Arduino s'effectue en fonction des besoins du projet.

L'ensemble des cartes Arduino se programment en C++ à l'aide d'un logiciel de programmation gratuit et open-source fourni par Arduino.

Le composant principal de la carte Arduino est un microcontrôleur Atmel. Un microcontrôleur est, en fait, un ordinateur complet sur une seule puce. En voici les principaux éléments :

- Le processeur ou CPU,
- La mémoire de type Flash qui héberge le programme à exécuter,
- La mémoire vive qui stocke les données variables,
- Les périphériques intégrés au composant,
- Les ports d'entrée/sortie qui connectent le microcontrôleur avec le monde extérieur.

Au final, le microcontrôleur n'est pas si différent d'un PC.

La carte Arduino intègre le microcontrôleur Atmel mais aussi des composants périphériques :

- Un port USB et de son contrôleur permettant de la connecter à un ordinateur,
- Un oscillateur permettant de cadencer le fonctionnement du microcontrôleur et de ces périphériques,
- Un régulateur de tension. La carte peut être alimentée par le port USB ou une alimentation dédiée,
- Des connecteurs d'entrée/sortie. L'agencement des connecteurs ainsi que leurs fonctionnalités sont identiques sur la plupart des cartes, permettant ainsi d'utiliser des modules additionnels *Shields*
- Différentes LED d'état (Power,RX,TX,….etc).
- Un bouton Reset [12].

3.3.1. Le principe du fonctionnement des cartes Arduino

La figure suivante illustre le synoptique descriptif du fonctionnement de la carte ARDUINO MEGA [13].

Figure 3. 1: principe de fonctionnement.

3.3.2. Les types d'Arduino

Il existe plusieurs cartes Arduino, chacune ayant ses caractéristiques.

Les différentes cartes Arduino [14] :

Arduino	Processeur	Flash ko	EEPROM ko	SRAM ko	Broches d'E/S numériques	...avec PWM	Broches d'entrée analogique	Type d'interface USB	Dimensions pouces	Dimensions mm
Diecimila	ATmega168	16	0,5	1	14	6	6	FTDI	2,7" x 2,1"	68,6 mm x 53,3 mm
Duemilanove	ATmega168/328P	16/32	0,5/1	1/2	14	6	6	FTDI	2,7" x 2,1"	68,6 mm x 53,3 mm
Uno	ATmega328P	32	1	2	14	6	6	ATmega8U2	2,7" x 2,1"	68,6 mm x 53,3 mm
Leonardo	ATmega32U4	32	1	2,5	20	7	12	ATmega32U4	2,7" x 2,1"	68,6 mm x 53,3 mm
Mega	ATmega1280	128	4	8	54	15	16	FTDI	4" x 2,1"	101,6 mm x 53,3 mm
Mega2560	ATmega2560	256	4	8	54	15	16	ATmega8U2	4" x 2,1"	101,6 mm x 53,3 mm
Due	Atmel SAM3X8E	512	0	96	54	12	12	SAM3X8E (USB Host), ATmega16u2 (programmation)	4" x 2,1"	101,6 mm x 53,3 mm
Fio	ATmega328P	32	1	2	14	6	8	Aucune	1,6" x 1,1"	40,6 mm x 27,9 mm
Nano	ATmega168 or ATmega328	16/32	0,5/1	1/2	14	6	8	FTDI	1,70" x 0,73"	43 mm x 18 mm
LilyPad	ATmega168V or ATmega328V	16	0,5	1	14	6	6	Aucune	2" ø	50 mm ø
Esplora	ATmega32U4	32	1	2,5	N/A	N/A	N/A	ATmega32U4	6,5" x 2,4"	165,1 mm x 60,96 mm

Tableau 3. 1 : Fiche technique des différentes cartes ARDUINO

3.4. Arduino mega2560

3.4.1. Spécification techniques de la carte Arduino Mega2560

Le Mega 2560 est une carte microcontrôleur basée sur l'ATmega2560. Il dispose de 54 broches numériques d'entrée / sortie (dont 15 peuvent être utilisées comme sorties

PWM), 16 entrées analogiques, 4 UART (ports série de matériel), un cristal oscillateur 16 MHz, une connexion USB, une prise d'alimentation, une embase ICSP et un bouton de remise à zéro.

Il contient tout le nécessaire pour soutenir le microcontrôleur; simplement le connecter à un ordinateur avec un câble USB ou de la puissance avec un adaptateur ou d'une batterie AC-DC pour commencer. La carte Mega 2560 est compatible avec la plupart des blindages conçus pour la Uno et les anciens conseils Duemilanove ou Diecimila.

Le Mega 2560 est une mise à jour du Mega Arduino , qu'il remplace [15].

3.4.2. Fiche technique

Microcontrôleur	ATmega2560
Tension de fonctionnement	5V
Tension d'entrée (recommandé)	7-12V
Tension d'entrée (limite)	6-20V
E / S numériques Pins	54 (dont 15 fournissent la sortie PWM)
Pins d'entrée analogique	16
DC Courant par I O Pin /	20 mA
Courant DC pour 3.3V Pin	50 mA
Mémoire flash	256 KB dont 8 KB utilisé par bootloader
SRAM	8 KB
EEPROM	4 KB
Vitesse de l'horloge	16 MHz
Longueur	101.52 mm
Largeur	53,3 mm
Poids	37 g

Tableau 3. 2 : Fiche technique carte Arduino Mega [15]

a. **Avertissement**

Le Mega 2560 a une poly fuse réinitialisable qui protège les ports USB de l'ordinateur de shorts et de surintensité. Bien que la plupart des ordinateurs fournissent leur propre protection interne, le fusible fournit une couche supplémentaire de protection. Si plus de

500 mA sont appliqués sur le port USB, le fusible cassera automatiquement la connexion jusqu'à ce court ou surcharge est supprimée [15].

3.4.3. Alimentation

La carte Arduino Uno peut-être alimentée via la connexion USB (qui fournit 5V jusqu'à 500mA) ou à l'aide d'une alimentation externe. La source d'alimentation est sélectionnée automatiquement par la carte.

L'alimentation externe (non-USB) peut être soit un adaptateur secteur (pouvant fournir typiquement de 3V à 12V sous 500 mA) ou des piles (ou des accus). L'adaptateur secteur peut être connecté en branchant une prise 2.1mm positif au centre dans le connecteur jack de la carte. Les fils en provenance d'un bloc de piles ou d'accus peuvent être insérés dans les connecteurs des broches de la carte appelés Gnd (masse ou 0V) et Vin (Tension positive en entrée) du connecteur d'alimentation.

La carte peut fonctionner avec une alimentation externe de 6 à 20 volts. Cependant, si la carte est alimentée avec moins de 7V, la broche 5V pourrait fournir moins de 5V et la carte pourrait être instable. Si on utilise plus de 12V, le régulateur de tension de la carte pourrait chauffer et endommager la carte. Aussi, la plage idéale recommandée pour alimenter la carte Mega2560 est entre 7 et 12 V [16].

a. **Les broches d'alimentation sont les suivants**

- VIN. La tension d'entrée positive lorsque la carte Arduino est utilisée avec une source de tension externe (à distinguer du 5V de la connexion USB ou autre source 5V régulée). On peut alimenter la carte à l'aide de cette broche ou, si l'alimentation est fournie par le jack d'alimentation, accéder à la tension d'alimentation sur cette broche.

 5V La tension régulée utilisée pour faire fonctionner le microcontrôleur et les autres composants de la carte. Le 5V régulé fourni par cette broche peut donc provenir soit de la tension d'alimentation VIN via le régulateur de la carte, ou bien de la connexion USB (qui fournit du 5V régulé) ou de tout autre source d'alimentation régulée.

 3.3V Une alimentation de 3.3V fournie par le circuit intégré FTDI (circuit intégré faisant l'adaptation du signal entre le port USB de l'ordinateur et le port série de l'ATmega) de la carte est disponible : ceci est intéressant pour certains circuits externes nécessitant cette tension au lieu du 5V. L'intensité maximale disponible sur cette broche est de 50 mA

- GND. Broche de masse (ou 0 V)

- IOREF. Cette broche sur la carte fournit la référence de tension avec laquelle le microcontrôleur fonctionne. Un écran correctement configuré peut lire la tension de la broche IOREF et sélectionner la source d'alimentation appropriée ou activer des traducteurs de tension sur les sorties pour travailler avec le 5 V ou 3,3V [16].

3.4.4. Mémoire

Le ATmega2560 a 256 Ko de mémoire flash pour le stockage du programme (dont 8 Ko sont utilisés pour le bootloader), Le ATmega2560 a également 8 Ko de SRAM (volatile) et 4 Ko de mémoire EEPROM (non volatile, qui peut être lue à l'aide de la librairie).

Le **bootloader** est un programme préprogrammé une fois pour toutes dans l'ATméga et qui permet la communication entre l'ATmega et le logiciel Arduino via le port USB, notamment lors de chaque programmation de la carte [16].

3.4.5. Entrée et sortie

Chacune des 54 broches numériques sur le Mega peut être utilisé comme une entrée ou une sortie, en utilisant les fonctions pinMode () , digitalWrite () , et digitalRead ().
Elles fonctionnent à 5 volts. Chaque broche peut fournir ou recevoir 20 mA en état de fonctionnement recommandé et a une résistance pull-up interne (déconnecté par défaut) de 20-50 k ohm. Un maximum de 40mA est la valeur qui ne doit pas être dépassée pour éviter des dommages permanents au microcontrôleur [16].

De plus, certaines broches ont des fonctions spécialisées :

- **Communication Série: 0 (RX) et 1 (TX); Série 1: 19 (RX) et 18 (TX); Série 2: 17 (RX) et 16 (TX); Série 3: 15 (RX) et 14 (TX).** Permet de recevoir **(RX)** et transmettre **(TX)** TTL données série. Pins 0 et 1 sont également connectés aux broches correspondantes de l'USB-TTL puce Serial ATmega16U2.
- **Interruptions externe : 2 (interruption 0), 3 (interruption 1), 18 (interruption 5), 19 (interruption 4), 20 (interruption 3), et 21 (interruption 2).** Ces broches peuvent être configurées de manière à déclencher une interruption sur un niveau bas, un front montant ou descendant, ou un changement de niveau. A l'aide de la fonction attachInterrupt ()

- **Impulsion PWM (largeur d'impulsion modulée): 2 à 13 et 44 à 46**. Fournir sortie PWM 8 bits avec le analogWrite () fonction.
- **SPI(interface série périphérique)**:les broches **50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** Ces broches supportent la communication SPI en utilisant la bibliothèque SPI . Les broches SPI sont également connectées sur le connecteur ICSP, qui est physiquement compatible avec les cartes Arduino / Genuino Uno et les anciennes cartes Duemilanove et Diecimila Arduino.
- **LED: 13**. Il est équipé d'un LED connecté à la broche numérique **13**. Lorsque la broche est la valeur HIGH, la LED est allumée, lorsque la broche est faible, il est hors tension.
- **TWI**: **20 (SDA) et 21 (SCL).** Soutien communication TWI en utilisant la bibliothèque de fil . Notez que ces broches ne sont pas dans le même emplacement que les broches TWI sur les vieilles planches Duemilanove ou Diecimila Arduino.
- Le Mega 2560 dispose de 16 entrées analogiques, chacune pouvant fournir une mesure d'une résolution de 10 bits (c.-à-d. sur 1024 niveaux soit de 0 à 1023) à l'aide de la très utile fonction analogRead() du langage Arduino. Par défaut, ces broches mesurent entre le 0V (valeur 0) et le 5V (valeur 1023), mais il est possible de modifier la référence supérieure de la plage de mesure en utilisant la broche AREF et l'instruction analogReference() du langage Arduino.
- **Remarque** : les broches analogiques peuvent être utilisées en tant que broches numériques.
- **Il y a quelques autres broches de la carte :**

AREF. Tension de référence pour les entrées analogiques. Utilisée avec l'instruction analogReference ().

Reset : Mettre cette broche au niveau LOW (bas) entraine la réinitialisation (le redémarrage) du microcontrôleur [16].

3.4.6. La communication

Le Mega 2560 conseil dispose d'un certain nombre de moyens pour communiquer avec un ordinateur, un autre conseil ou d'autres microcontrôleurs. Le ATmega2560 fournit quatre UART (Universal Asynchronous Receiever Transmitter) pour communication série de niveau TTL (5V) et qui est disponible sur les broches 0 (RX) et 1 (TX). Un circuit intégré ATmega16U2 sur la carte assure la connexion entre cette communication

série vers le port USB de l'ordinateur et apparaît comme un port COM virtuel pour les logiciels de l'ordinateur.

Le code utilisé pour programmer l'ATmega16U2 utilise le driver standard USB COM, et aucun autre driver externe n'est nécessaire. Cependant, sous Windows, un fichier .inf est requis.

Le logiciel Arduino inclut une fenêtre terminal série (ou moniteur série) sur l'ordinateur et qui permet d'envoyer des textes simples depuis et vers la carte Arduino. Les LEDs RX et TX sur la carte clignotent lorsque les données sont transmises via le ATmega8U2 / ATmega16U2 et le circuit intégré USB-vers-série et la connexion USB vers l'ordinateur (mais pas pour les communications série sur les broches 0 et 1).

Une librairie Série Logicielle permet également la communication série (limitée cependant) sur n'importe quelle broche numérique de la carte Mega2560.

Le Mega 2560 supporte également la communication par protocole I2C (ou interface **TWI** (Two Wire Interface - Interface "2 fils") et **SPI** :

 Le logiciel Arduino inclut la librairie Wire qui simplifie l'utilisation du bus I2C.

 Pour utiliser la communication SPI (Interface Série Périphérique), la librairie pour communication SPI est disponible.

Rappel :

Le Mega 2560 n'utilise pas la puce du pilote FTDI USB dans les conceptions antérieures. Au lieu de cela, il dispose de la ATmega16U2 (ATmega8U2 dans les conseils de révision 1 et révision 2 Arduino) programmée comme un convertisseur USB-série.

Révision 2 de la Mega 2560 conseil a une résistance en tirant la ligne 8U2 HWB au sol, ce qui rend plus facile à mettre en mode DFU.

Révision 3 de la carte Arduino et le courant Genuino Mega 2560 ont les caractéristiques améliorées suivantes:

- **brochage:** SDA et SCL pins - près de la broche AREF - et deux autres nouvelles broches placées à proximité de la broche RESET, l'IOREF qui permettent aux boucliers l'adaptation à la tension fournie par le conseil d'administration. A l'avenir, les boucliers seront compatibles à la fois avec le conseil d'administration qui utilise l'AVR fonctionnant avec 5V et avec le conseil d'administration qui utilise ATSAM3X8E et

fonctionnant avec 3.3V. Le second est une broche non connectée, réservée pour des fins futures.

- Stronger circuit RESET.
- Atmega 16U2 remplace le 8U2 [16].

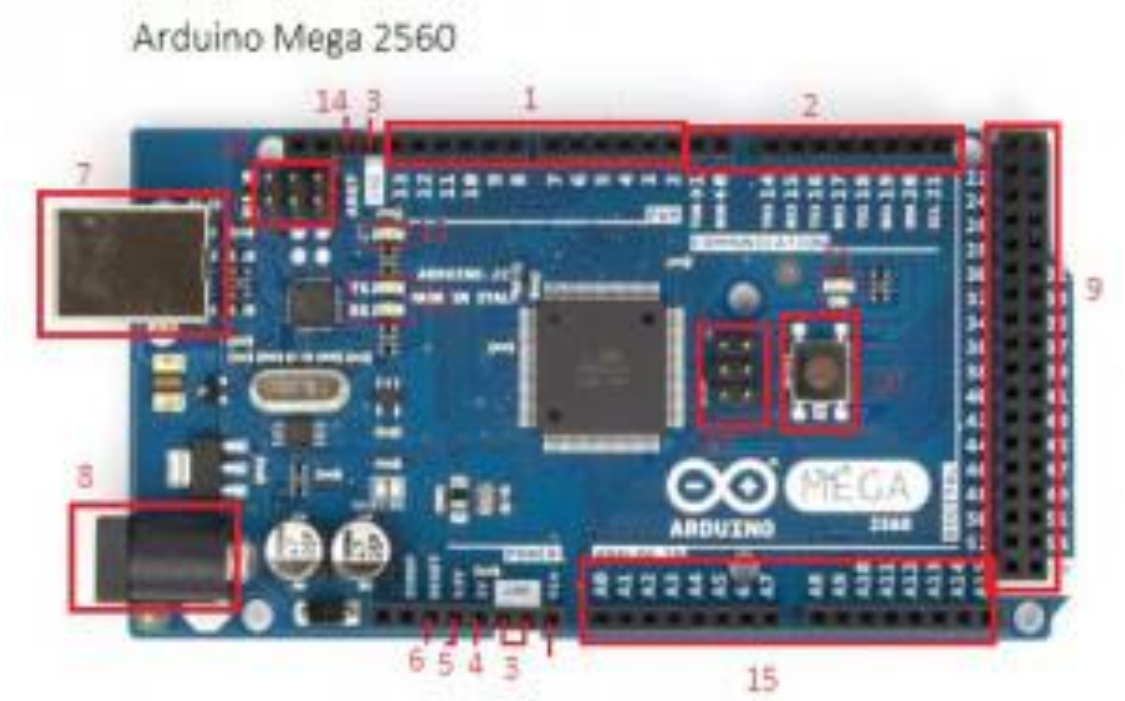


Figure 3. 2 : carte Arduino Mega [17]

Arduino Mega 2560 :

1. Les ports **PWM**.
2. Les ports de **communication**.
3. Le **ground**.
4. Sortie **5v**.
5. Sortie **3,3v**.
6. Entrée **reset**.
7. Port **série**.
8. Prise d'**alimentation**.
9. Pins **digital**
10. Bouton **reset**. Sert à redémarrer la carte pour relancer le programme.
11. LED verte qui montre que la carte est allumée.
12. LEDs jaunes qui indiquent l'utilisation du port série

13. LED jaune qui indique l'activité du processeur.

14. **AREF**.

15. Les pins d'entrée **analogiques**.

16. Une prise **ICSP** pouvant servir à télécharger un autre bootloader (ce qui lance le programme).

3.5. Le côté logiciel

3.5.1. La programmation

Le Mega 2560 bord peut être programmé avec Arduino (IDE) .

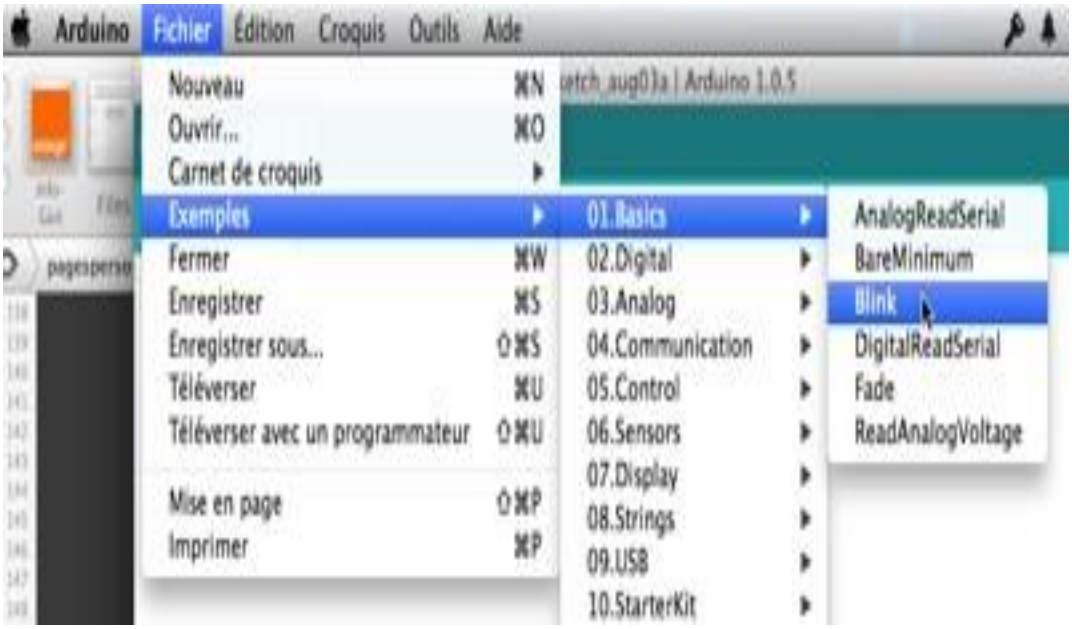
Les ATmega2560 sur le Mega 2560 sont préprogrammés avec un bootloader qui permet de télécharger le nouveau code sans l'utilisation d'un programmeur de matériel externe. Il communique en utilisant le protocole original STK500.

On peut également contourner le bootloader et programmer le microcontrôleur à travers le ICSP (In-Circuit Serial Programmation) en utilisant Arduino ISP ou similaire

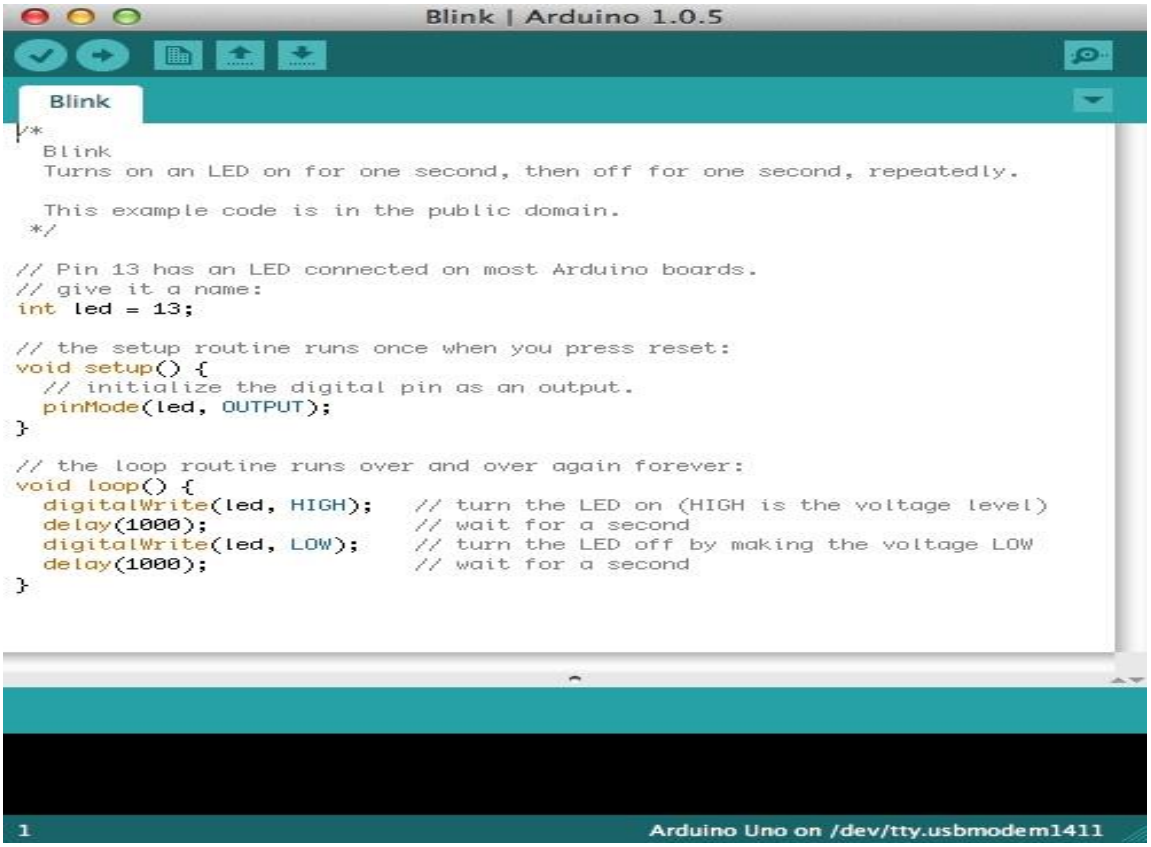
Arduino se compose de deux parties : une partie matérielle, représentée par la carte Arduino elle-même et une partie logicielle téléchargeable à installer (disponible pour les plateformes Windows, Mac OSX et Linux) permettant d'écrire des croquis (Sketches en anglais) et les envoyer au microcontrôleur de la carte Arduino.

IDE Arduino est un environnement intégré (IDE), Il permet la saisie du code mais également sa transformation en un langage que l'Arduino peut comprendre (compilation). L'environnement permettra ensuite de transférer le croquis vers la carte (Téléverser). Après avoir téléchargé et installé le logiciel (suivre simplement les différentes étapes), nous allons vérifier que l'ordinateur communique bien avec la carte en uploadant (téléversant) un croquis.

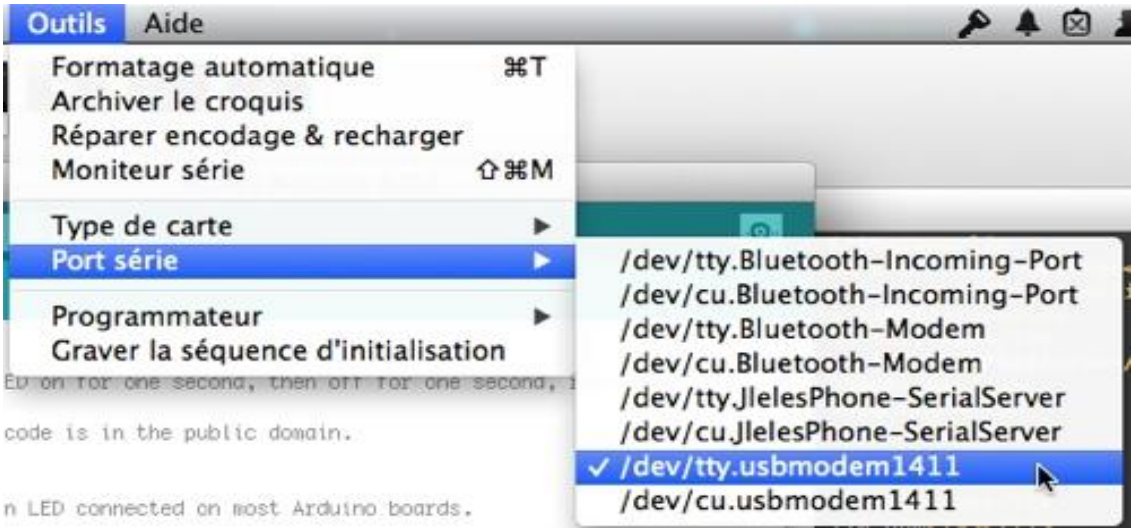
- Connecter le câble USB entre la carte Arduino et un port USB de l'ordinateur.
- Exécuter l'application Arduino
- Connecter la carte à l'ordinateur avec le câble USB fourni.
- Choisir le croquis Blink disponible à Fichier > Exemples > O1.Basics > Blink [16].



- Le code source suivant se charge,



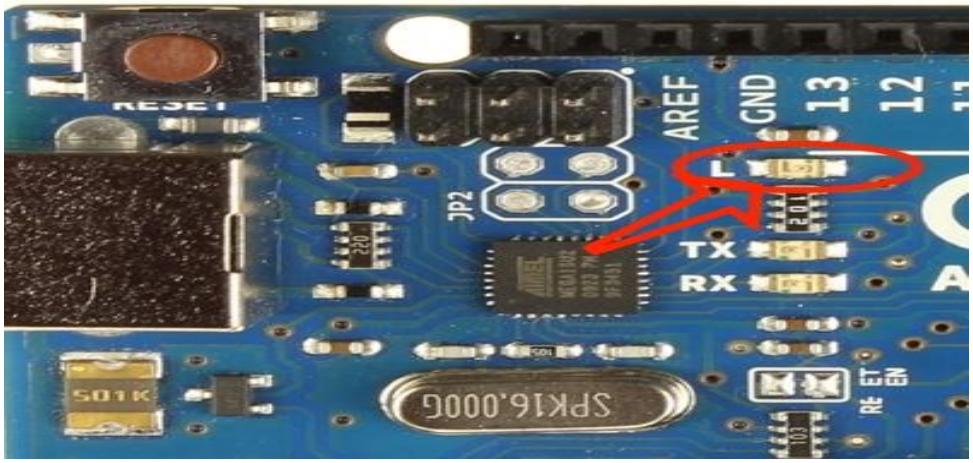
- Avant de téléverser, ce croquis il faudra, au préalable, choisir le port série qui communiquera avec cette dernière ; sous MAC, il sera de la forme /dev/tty.usbmodemxxxx



- Pour téléverser le croquis vers la carte Arduino, cliquer sur le bouton Téléverser .

.

- Une barre de progression indiquera l'état d'avancement. Les LEDs TX et RX de la carte vont clignoter indiquant ainsi une communication entre l'ordinateur et la carte. Et si tout c'est bien passé, la LED identifiée ***L*** se mettra à clignoter.



a. **Les bases de la programmation sous Arduino**

Utilisant le langage C. Nous allons nous concentrer sur les spécificités propres à Arduino en considérant que les bases du langage C sont acquises.

```
/*
  Blink
 // Turns on an LED on for one second, then off for one second, repeatedly.
//  This example code is in the public domain.
 */
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

b. **la structure d'un programme**

Un programme Arduino comporte trois parties :

1. la partie déclaration des variables (optionnelle)
2. la partie initialisation et configuration des entrées/sorties : la fonction **setup ()**
3. la partie principale qui s'exécute en boucle : la fonction **loop ()**

Dans chaque partie d'un programme, sont utilisées différentes instructions issues de la syntaxe du langage Arduino [18].

Setup () contient des lignes de programmes qui ne seront exécutées qu'une seule fois lorsque l'Arduino sera mis sous tension ou redémarré. Setup signifie configuration et c'est justement l'objet de cette fonction, configurer des éléments spécifiques et définir la configuration. Voyez cela comme le code de démarrage.

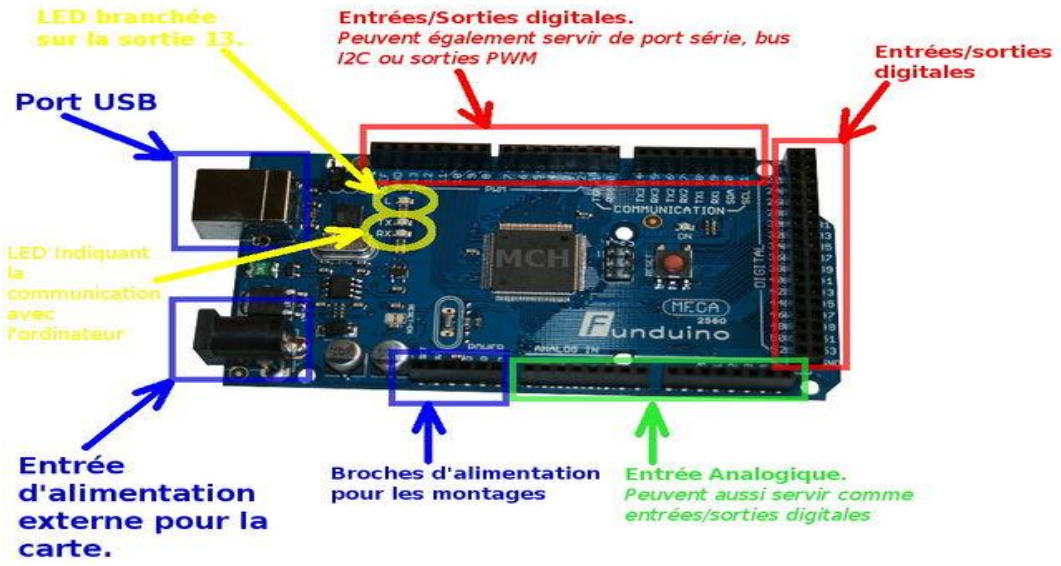


Figure 3. 3: Détail d'une carte Arduino Mega réel [19]

Ex : Utiliser les ports d'E/S numériques

Notre carte Arduino Mega2560 comporte une cinquantaine de ports numériques. Lorsque nous utiliserons un ou plusieurs de ces ports, il faudra préalablement indiquer au microcontrôleur que nous souhaitons les utiliser en entrée ou en sortie. Pour cela, il faut placer l'instruction **pinMode** dans la fonction setup(). Il est possible d'utiliser, dans le cas d'un port configuré en Entrée, une résistance interne de *pullup* en utilisant le paramètre **INPUT_PULLUP**. Par défaut, cette fonctionnalité n'est pas active ; il faut l'indiquer explicitement.

```
// the setup routine runs once when you press reset:

void setup() {

   // initialize the digital pin 2 as an output.

  pinMode(2, OUTPUT);

   // initialize the digital pin 3 as an input.

   pinMode(3, INPUT);

   // initialize the digital pin 4 as an input with pullup resistor.

   pinMode(4, INPUT_PULLUP);

}
```

On remarque que certains ports avaient le symbole tilde devant le numéro (comme par exemple le port ~3), cela concerne la gestion du **PWM** (Pulse-Width Modulation).

loop() Loop signifie boucle et le code qui sera placé dans cette fonction sera exécuté en boucle encore et encore. On peut parfaitement saisir ce croquis minimaliste, le compiler et le téléverser dans l'Arduino.

Ce programme minimaliste ne fait rien, ni au démarrage, ni ensuite, mais le fera en boucle et indéfiniment.

c. Les shields

Il existe de nombreux shields que l'on traduit parfois dans les documentations par «boucliers ». Le terme « extension » paraitrait plus approprié. Un « shield » Arduino est une petite carte qui se connecte sur une carte Arduino pour en augmenter ses fonctionnalités. Quelques exemples de « shields » : [18]

• Afficheur graphique
• Ethernet et carte SD
• GPS
• Carte de prototypage (type labdec)

Figure 3. 4: Arduino uno + shieled Ethernet

3.6. Les avantages

• Pas cher

• Environnement de programmation clair et simple.
• Multiplateforme : tourne sous Windows, Macintosh et Linux.
• Nombreuses librairies disponibles avec diverses fonctions implémentées.
• Logiciel et matériel open source et extensible.
• Nombreux conseils, tutoriaux et exemples en ligne (forums, site perso etc...)
• Existence de « shield » : ce sont des cartes supplémentaires qui se connectent sur le module Arduino pour augmenter les possibilités comme, par exemple : afficheur graphique couleur, interface Ethernet, GPS, etc...
Par sa simplicité d'utilisation, Arduino est utilisé dans beaucoup d'applications comme l'électronique industrielle et embarquée, le modélisme, la domotique mais aussi dans des domaines différents comme l'art contemporain ou le spectacle

3.7. Conclusion

Il y a de nombreux microcontrôleurs et de nombreuses plateformes basées sur des microcontrôleurs disponibles pour l'électronique programmée : Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, et beaucoup d'autres qui offrent des fonctionnalités comparables.
Tous ces outils prennent en charge les détails compliqués de la programmation des microcontrôleurs et les intègrent dans une présentation facile à utiliser.
De la même façon, le système Arduino simplifie la façon de travailler avec les microcontrôleurs tout en offrant plusieurs avantages pour les enseignants, les étudiants et les amateurs intéressés par les autres systèmes.

4.1Etude de différentes parties du système

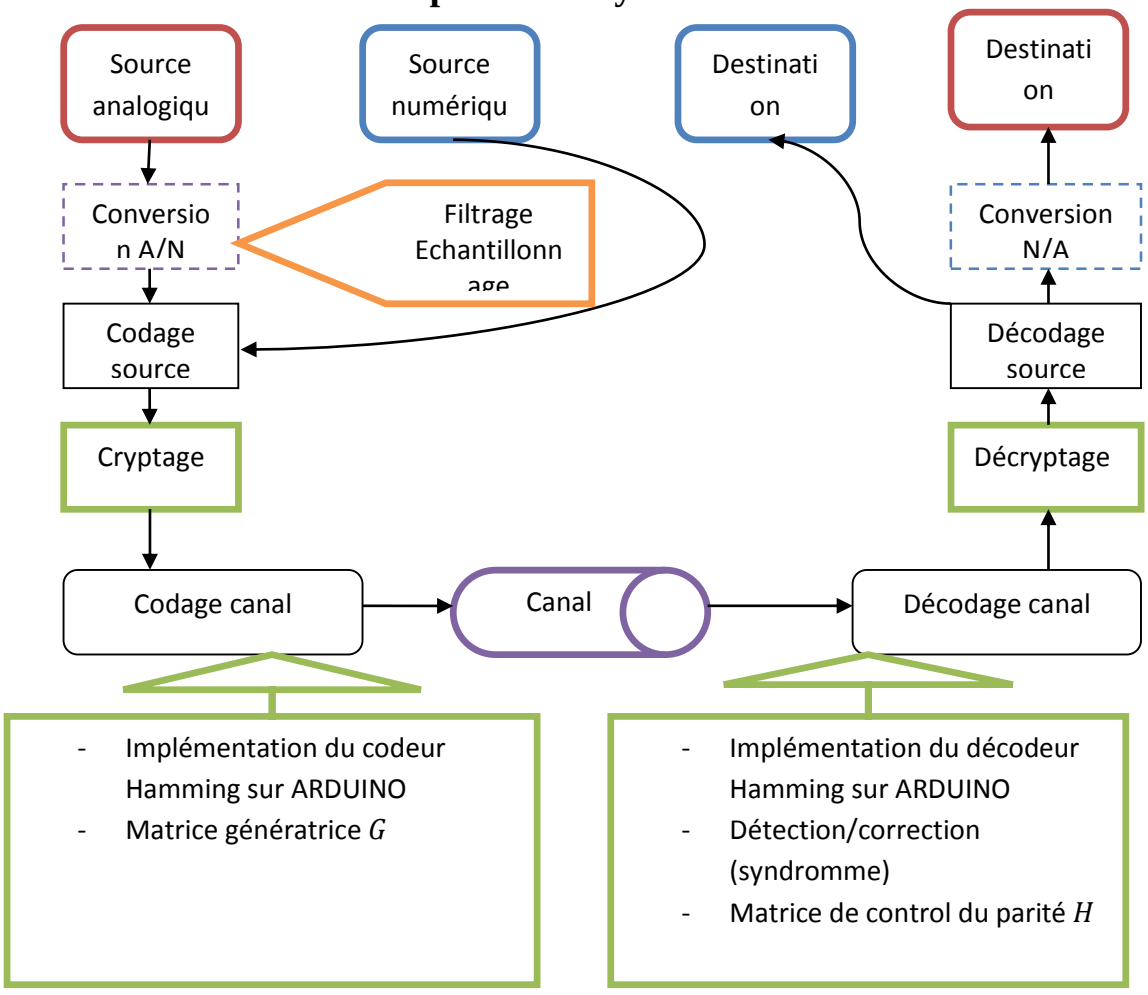


Figure 4. 1 : schéma simplifier du système à étudie

Dans notre projet, nous nous basons sur le codage canal. Dans ce principe, nous devons implémenter le codec Hamming utilisant une carte Arduino MegaA2560.

4.2. Codage canal

Dans cette partie, on a deux cas : l'information entrée qui est soit analogique, soit numérique

- Dans le cas où elle est numérique, on passe directement au codage.
- Dans le cas où elle est analogique, on doit numériser le signal par conversion A/N

Le signal que nous voulons transmettre est un signal sinusoïdal. Son principal inconvénient est qu'il est continu (il est également très fortement redondant à cause de sa périodicité). Pour cela, une forme simple de compression est de ne prendre des valeurs que ponctuellement, c'est a dire, réaliser un échantillonnage. Nous remarquons ainsi que cette étape de codage se fait avec perte d'information. Autrement dit, les erreurs induites par ce codage introduisent du bruit dans le signal à transmettre. Cette phase est composée de deux parties essentielles :

4.2.1. *L'échantillonnage*

A chaque fois qu'une durée appelée période d'échantillonnage s'écoule, on retient la valeur numérique du signal qu'on échantillonne. Le signal obtenu est discrétisé et donc beaucoup moins riche en information, mais beaucoup plus facile à transmettre. Son amplitude, elle, n'a toujours pas changée.

Le signal va (t) est transformé en une suite discrète de valeurs correspondant aux différents échantillons.

va (t)$\rightarrow V * e(k.T_e)$

Te : période d'échantillonnage

4.2.2. *La quantification*

Après avoir échantillonné un signal, nous obtenons une série de mesures auquel le convertisseur analogique-numérique va attribuer une valeur numérique composée uniquement de 0et de 1 caractérisant l'intensité du signal analogique d'origine à l'instant de l'échantillon. Ce processus s'appelle la quantification. Le principal paramètre de la quantification est le choix du codage. Un signal peut être codé en un nombre de bits qui est une puissance de 2 (2;4; 8; ...) et représente le nombre de chiffres de la valeur d'un échantillon. L'utilisateur peut donc régler ce paramètre suivant qu'il veuille une excellente qualité ou au contraire une qualité moyenne. Chaque échantillon prélevé du

signal d'origine est donc caractérisé par une valeur numérique en bits composée de 0 et de 1 plus ou moins grande suivant le codage choisi par l'utilisateur

On fait passer le signal bloc filtre passe bas, puis un bloc échantillonneur/bloqueur comme le présente la figure suivante.

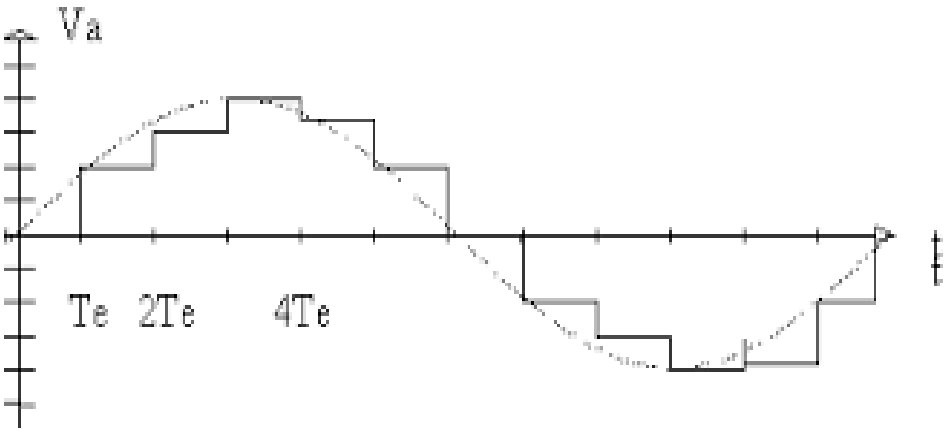


Figure 4. 2: L'échantillonnage et la quantification du signal

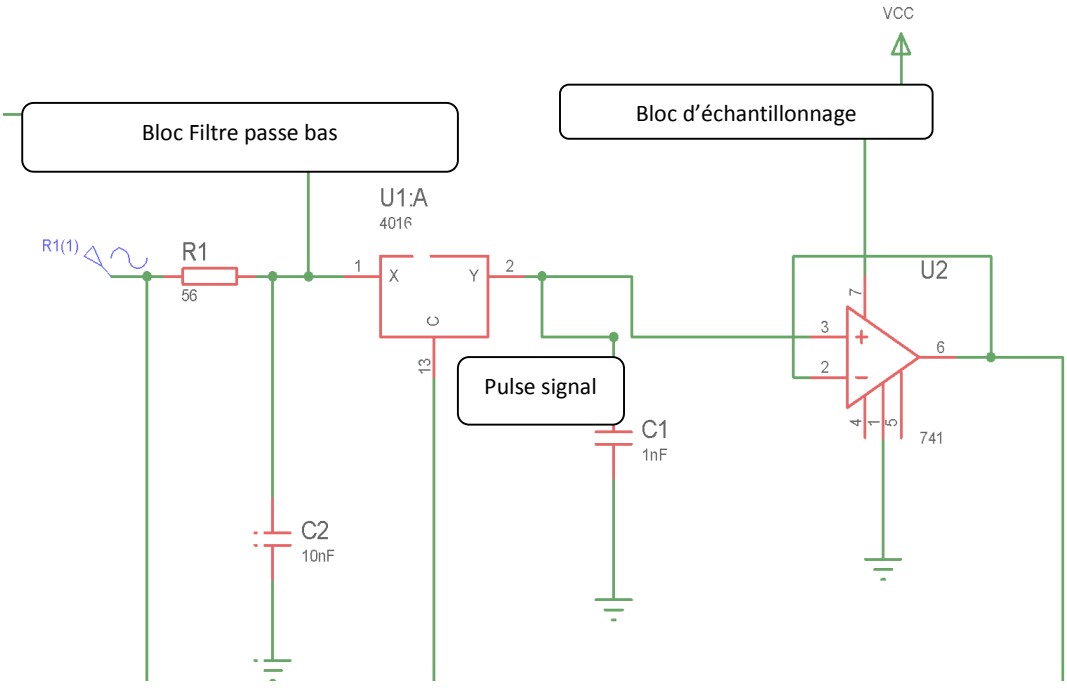


Figure 4. 3: bloc filtrage+bloc échantillonnage

4.3. La conversion A/N se fait à l'aide de l'ARDUINO

4.3.1. La carte ARDUINO à microcontrôleur

Un **microcontrôleur** (en notation abrégée **µC**) est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties. Les microcontrôleurs se caractérisent par un plus haut degré d'intégration, une plus faible consommation électrique, une vitesse de fonctionnement plus faible (de quelques mégahertz jusqu'à plus d'un gigahertz) et un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels.

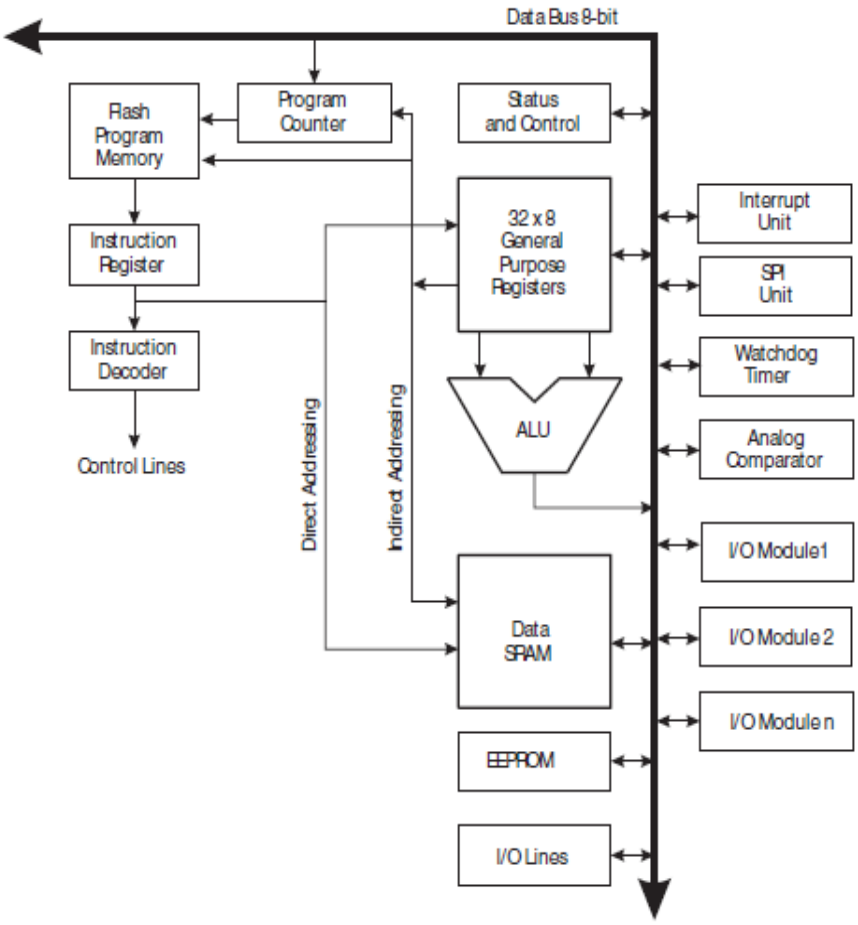


Figure 4. 4: Schéma de fonctionnement d'un µC [20]

La carte Arduino dispose, en plus du microcontrôleur :

- de 54 broches numériques d'entrées/sorties (+ 16 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée)),
- de 16 entrées analogiques (utilisables en broches entrées/sorties numériques),
- d'un quartz 16Mhz,
- d'une connexion USB,
- d'un connecteur d'alimentation jack,
- et d'un bouton de réinitialisation (reset).

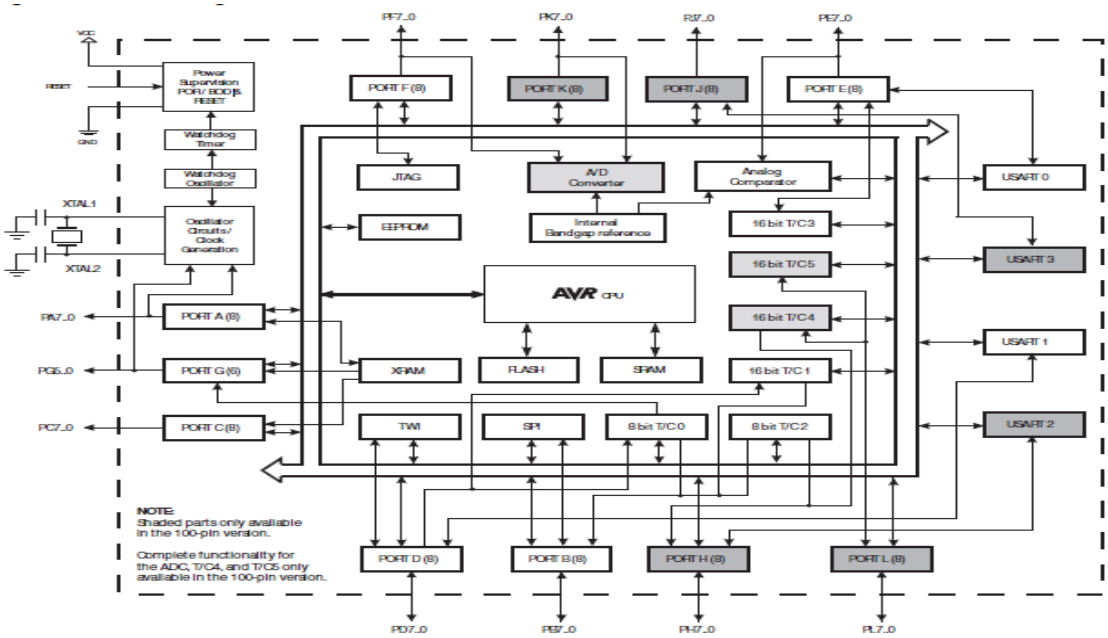


Figure 4. 5: le schéma interne d'une carte Arduino ATMega2560 [20]

La carte Arduino contient tout ce qui est nécessaire au fonctionnement du microcontrôleur. Pour pouvoir l'utiliser, il suffit simplement de la connecter à un ordinateur à l'aide d'un câble USB (ou de l'alimenter avec un adaptateur secteur ou une pile, mais ceci n'est pas indispensable, l'alimentation étant fournie par le port USB) ; puis de télécharger, dans sa mémoire, un programme machine exclusivement binaire que le µC exécute tant qu'il reste sous tension.

Convertisseur analogique-numérique : Le CAN contient un circuit d'échantillonnage et maintien qui assure que la tension d'entrée au CAN est maintenue à un niveau constant lors de la conversion

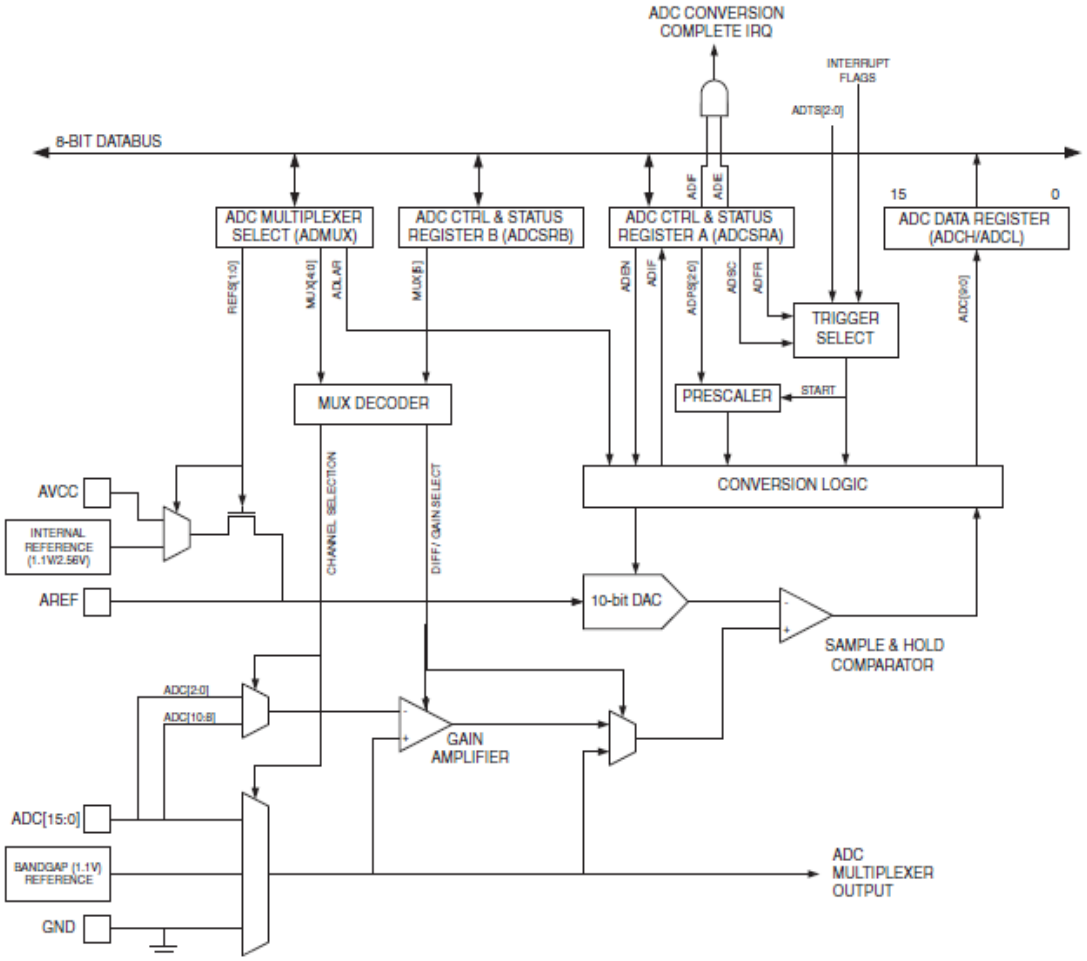


Figure 4. 6 : Schéma bloc Convertisseur analogique-numérique [20]

a. **Opération de conversion**

Le CAN convertit une tension d'entrée analogique en une valeur numérique de 10 bits par approximations successives. La valeur minimale représente GND et la valeur maximale représente la tension sur la broche AREF moins 1 LSB.

Le CAN génère un résultat de 10 bits qui est présenté dans le CAN Registres de données, ADCH et ADCL, par défaut.

4.4. Implémentation des entrées analogiques et numériques

Dans le premier cas (le mot est numérique), on utilise des switches pour introduire le mot. Pour cela, les switches sont reliés directement aux broches de l'Arduino (46-53). La figure 26 montre cette situation.

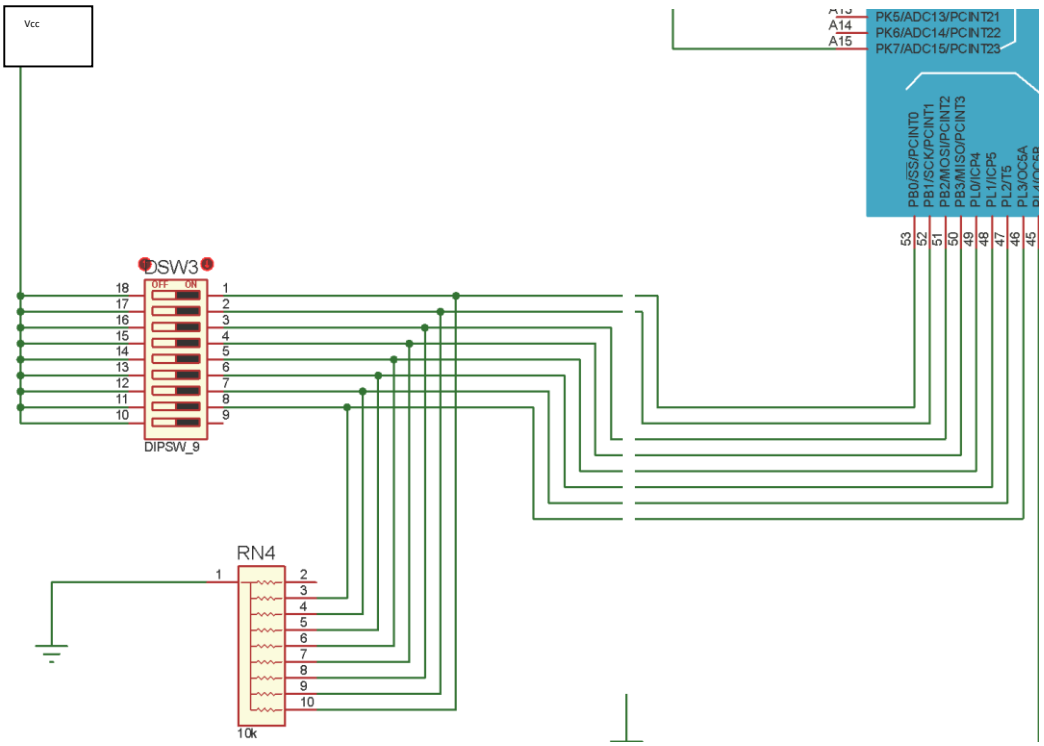


Figure 4. 7: Branchement des entrées numériques

Les switches sont alimentés par V_{cc}. A la sortie, on ajoute des résistances à mise à niveau puis on les branche aux pins de l'Arduino.

Dans le deuxième cas, où le signal est analogique, on relie la sortie de l'échantillonneur à une entrée analogique de l'Arduino (la broche A_{15}).

Simulation du mot codé et l'erreur

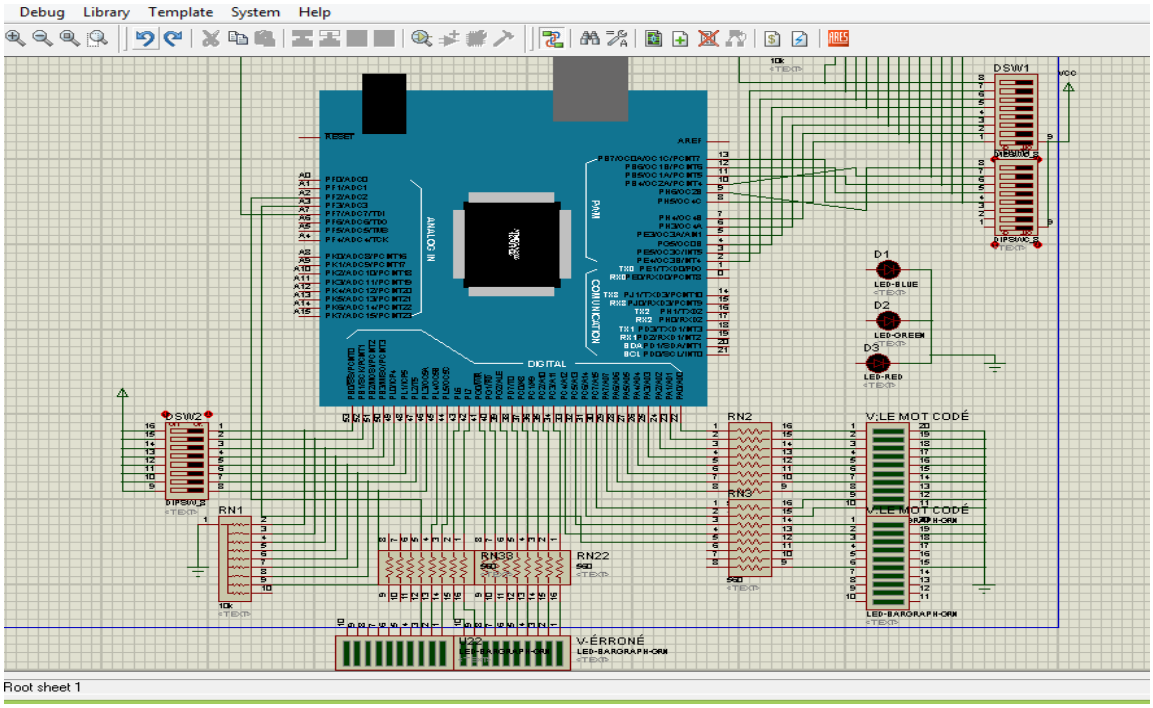


Figure 4. 8 : Simulation du mot codé et l'erreur

Aucune erreur appliquée

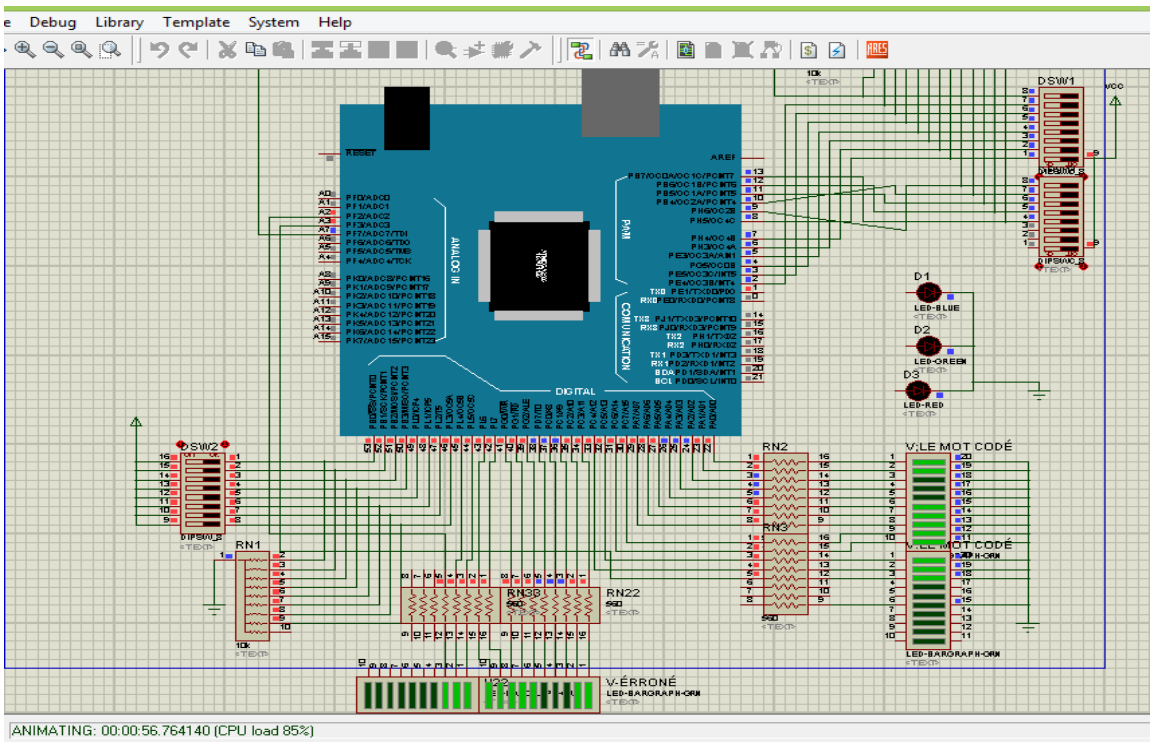


Figure 4. 9: Aucune erreur appliquée

On applique une erreur sur les switch (1-7)

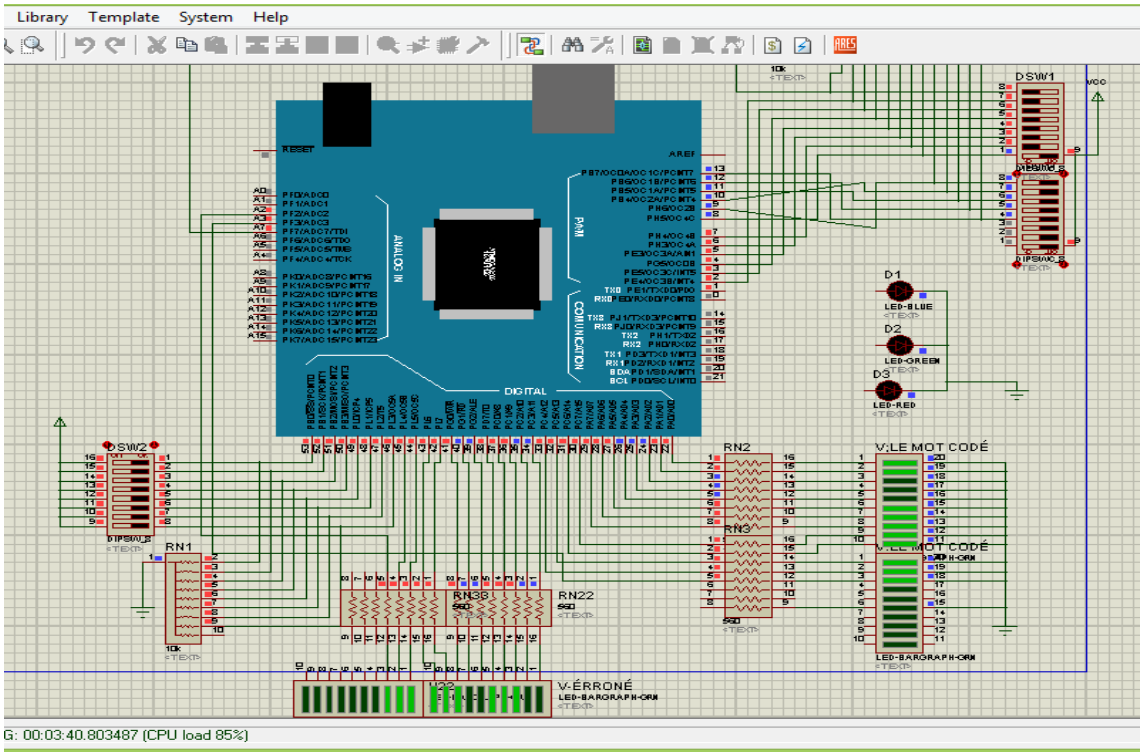


Figure 4. 10: On applique une erreur sur les switch (1-7)

On applique une erreur sur les switch (1-8)

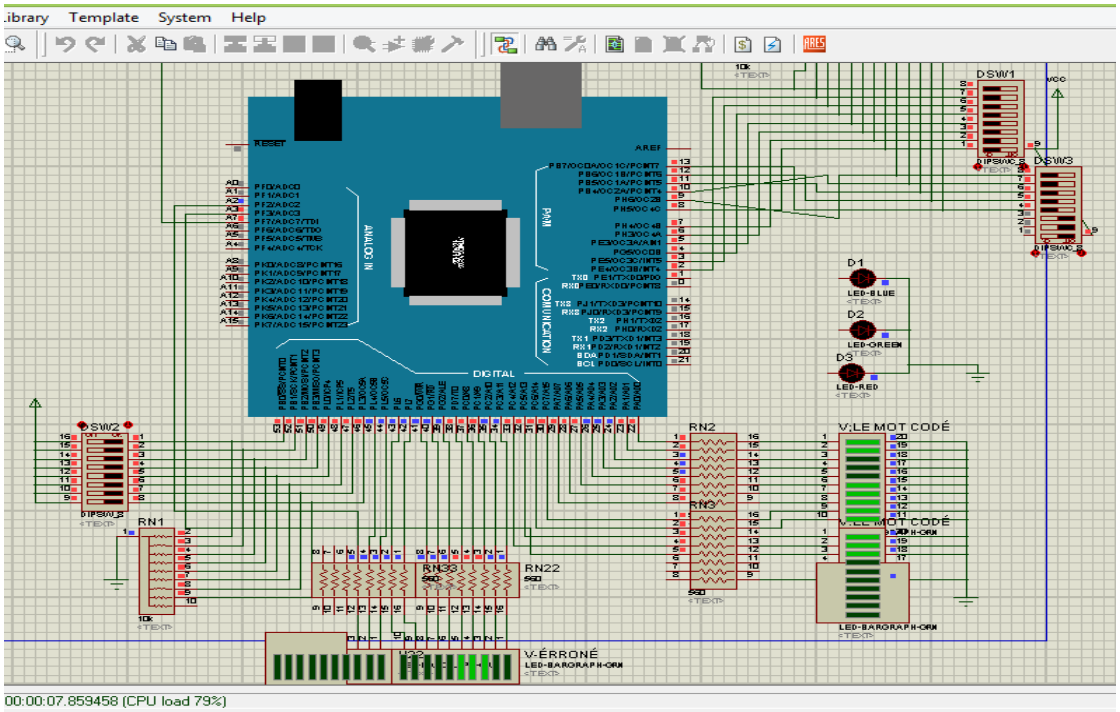


Figure 4. 11: On applique une erreur sur les switch (1-8)

On va éteindre les switches du mot codé

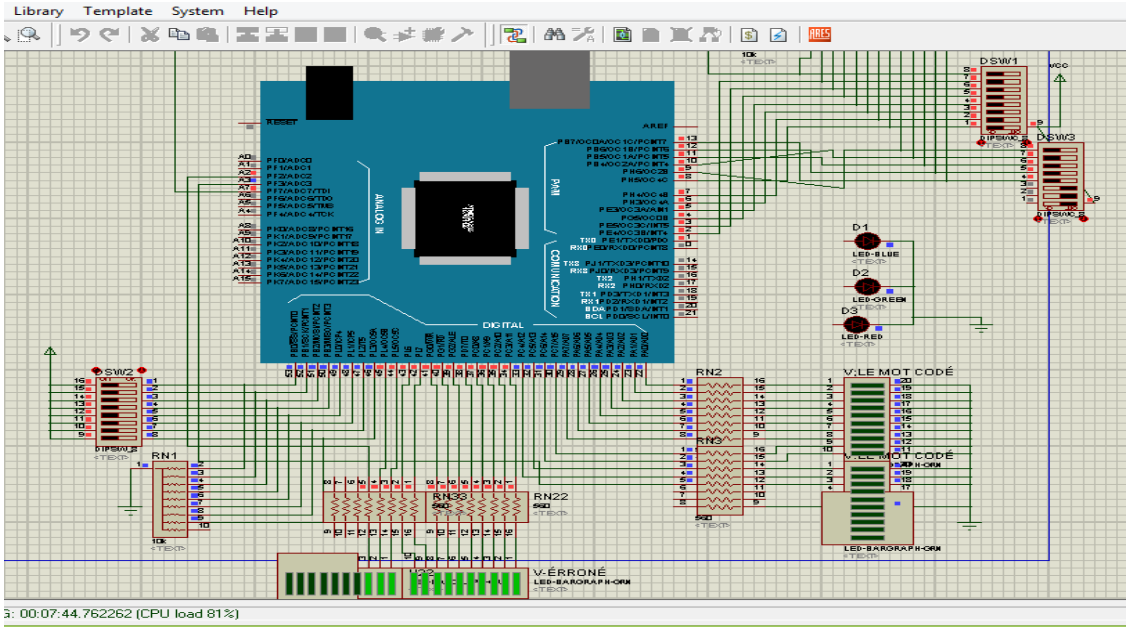


Figure 4. 12: On va éteindre les swtiches du mot codé

4.5. Implantation de l'afficheur LCD

Les afficheurs à cristaux liquides, appelés afficheurs LCD (Liquid Crystal Display), sont des modules compacts intelligents et nécessitent peu de composants externes pour un bon fonctionnement. Leur consommation en courant est faible, de l'ordre de $1mA$ à $5\ mA$.

Plusieurs afficheurs sont disponibles sur le marché et diffèrent les uns des autres par leurs dimensions (de 1 à 4 lignes de 6 à 80 caractères) et aussi par leurs caractéristiques techniques et leur tension de service. Certains sont dotés d'un rétro-éclairage. Cette fonction fait appel à des LED montées derrière l'écran du module. Nous avons opté pour le mode 8 bits et utilisé un schéma classique pour l'implantation.

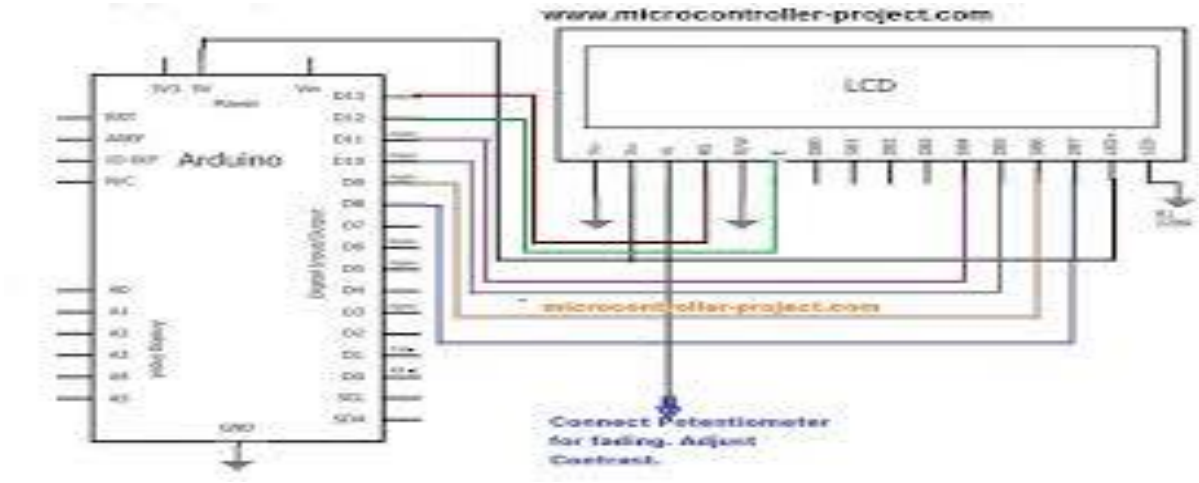


Figure 4. 13: Schéma d'implantation de l'afficheur LCD

4.5.1. *Schéma fonctionnel*

La figure suivante montre le schéma fonctionnel de l'afficheur LCD choisi

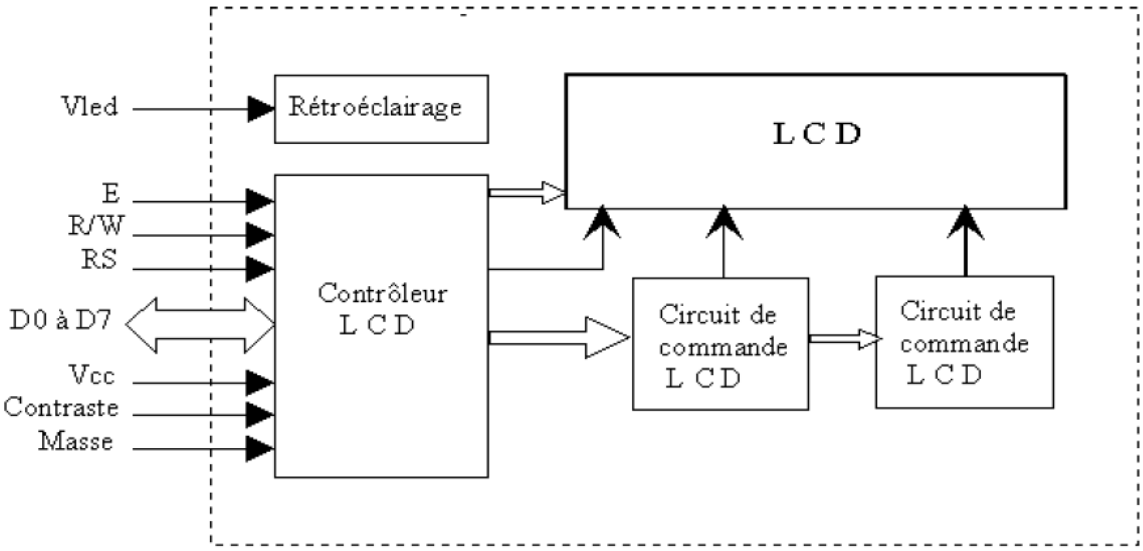


Figure 4. 14: Schéma fonctionnel d'un LCD

On constate que l'affichage comporte d'autres composants que l'afficheur à cristaux liquides (LCD) seul. Un circuit intégré de commande spécialisé, le LCD Controller, est chargé de la gestion du module. Le "contrôleur" remplit une double fonction : d'une part, il commande l'affichage et de l'autre, il se charge de la communication avec l'extérieur.

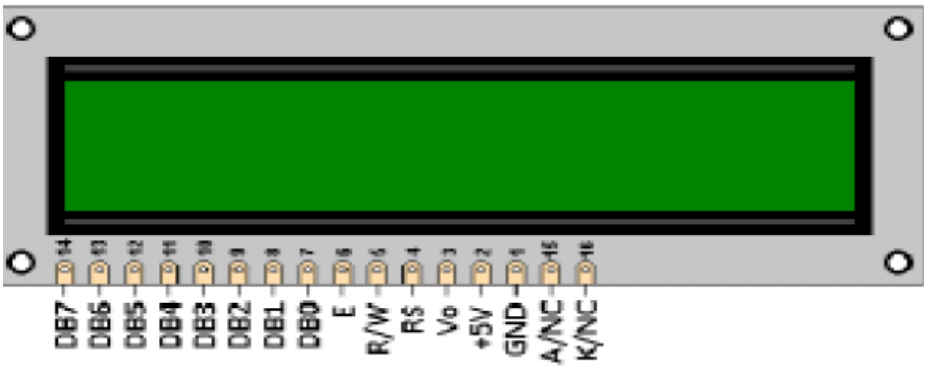


Figure 4. 15 : Photo d'un LCD et son brochage

Brochage de l'afficheur LCD

Broche	Nom	Niveau de tension	Fonction
1	V_{ss}	-	Masse
2	V_{dd}	-	Alimentation positive +5V
3	V_0	0 - 5V	Cette tension permet, en la faisant varier entre 0 et +5V, le réglage du contraste de l'afficheur.
4	RS	TTL	Sélection du registre (Register Select) Grâce à cette broche, l'afficheur est capable de faire la différence entre une commande et une donnée. Un niveau bas indique une commande et un niveau haut indique une donnée.
5	R/W	TTL	Lecture ou écriture (Read/Write) L : Écriture/H : Lecture
6	E	TTL	Entrée de validation (Enable) active sur front descendant. Le niveau haut doit être maintenu pendant au moins 450 ns à l'état haut.
7	D_0	TTL	
8	D_1	TTL	
9	D_2	TTL	
10	D_3	TTL	D0 –> D7 Bus de données bidirectionnel 3 états (haute impédance lorsque E=0)
11	D_4	TTL	
12	D_5	TTL	
13	D_6	TTL	
14	D_7	TTL	
15	A	-	Anode rétro éclairage (+5V)
16	K	-	Cathode rétro éclairage (masse)

Tableau 4. 1: Brochage d'un Afficheur LCD

Les broches 15 et 16 ne sont présentes que sur les afficheurs LCD avec retro-éclairage.

4.6. Mot de 8 bits introduit manuellement

Il consiste en un ensemble de 8 interrupteurs (micro-switches) numérotés de 1 à 8, chacun correspondant à un bit du mot digital introduit de façon répétitive sur le bus de données de l'émetteur.

Le mot numérique est donc constitué de 8 bits $C_8, C_7, C_6, C_5, C_4, C_3, C_2, C_1$

Le bus transmet ce signal numérique à l'entrée de l'étage d'encodage.

Exemple que nous allons traiter :

Un code c (n ,k) complètement spécifié par une matrice génératrice G de dimension $(K\ x\ n)$. La forme de cette matrice est la suivante :

$$G\ \ G= \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \\ g_8 \end{bmatrix} = \left[\begin{array}{cccc|cccccccc} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \qquad (4.1)$$

Matrice $P\ [n-k\ \times\ k]$ matrice identité $I\ [k\ \times\ k]$

$$G=[P\quad .\quad I_k] \qquad (4.2)$$

Le vecteur à code est : C : $(c_1\ c_2 c_3 c_4 c_5 c_6 c_7\ c_8)$

Déterminer le vecteur codé V :

Par définition, V=C.G =$[c_1\ c_2\ c_3 c_4\ c_5\ c_6\ c_7\ c_8]$

$$V=[c1\ c2\ c3\ c4\ c5\ c6\ c7\ c8\ ].$$

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.3)$$

$$V = \begin{bmatrix} c_1 + c_2 + c_4 + c_{5+}c_7 \\ c_1 + c_3 + c_4 + c_6 + c_7 \\ c_2 + c_3 + c_4 + c_8 \\ c_5 + c_6 + c_7 + c_8 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \end{bmatrix} \quad (4.4)$$

Format systématique d'un mot codé :

Partie des bits redondants	Partie du message
v_1 , v_2, v_3, v_4	$v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$

4.6.1. **Le codage**

C'est une technique pour la détection et la correction d'erreurs (EDC) consistant en l'ajout, pour chaque paquet de données, d'un ensemble de bits de contrôle.

Il existe plusieurs types de méthodes de codage de blocs. Celle qui est utilisée dans ce banc d'essai est très répandue et est appelée la technique de codage de HAMMING. Cette procédure consiste en l'ajout pour chaque mot de 8 bits original $(C_8, C_7, C_6, C_5, C_4, C_3, C_2, C_1)$ de quatre de contrôle (r_4, r_3, r_2, r_1) de sorte que le nouveau mot devienne ($r_4, r_3, r_2, r_1, C_8, C_7, C_6, C_5, C, C_3, C_2, C_1$) Les bits de parité sont générés en utilisant l'algorithme suivant :

$r_1 = r_3 \oplus C_2 \oplus C_4 \oplus C_5 \oplus C_7$

$r_2 = C_1 \oplus OR C_3 \oplus C_4 \oplus C_6 \oplus C_6$

$r_3 = C_2 \oplus C_3 \oplus C_3 \oplus C_8$

$r_4 = C_5 \oplus C_6 \oplus C_7 \oplus C_8$

Où $\oplus$ est l'opérateur ou exclusif entre deux bits. L'opération XOR est une addition modulo 2 et sa table de vérité est comme suit :

C_1	C_2	$C_1 \oplus C_2$
0	0	0
0	1	1
1	0	1
1	1	0

Tableau 4. 2: tableau de l'opération $\boldsymbol{xor}$

Les 12 bits du mot $r_4, r_3, r_2, r_1, C_8, C_7, C_6, C_5, C, C_3, C_2, C_1$sont transmis en série au récepteur.

4.6.2. **Contrôle de parité**

L'on ajoute, pour chaque paquet de données transmis ou mot, un bit dont la valeur est telle que le nombre de 1 devient pair P (contrôle de parité paire) ou impair (contrôle de parité impaire).

Le récepteur aura simplement à compter le nombre de 1 reçu pour chaque mot pour détecter si celui-ci contient une erreur ou non.

Le contrôle de parité n'est efficace que si l'on a une erreur sur un seul bit dans un mot.

Le vecteur V de mot reçu avec le bit de parité :

$$V=[P\ \ C_8\ \ C_7\ C_6\ \ C_5\ \ r_4\ C_4\ C_3\ C_2\ r_3\ C_1\ r_2\ r_1] \quad (4.6)$$

Matrice de contrôle de parité H du code :

V.H^T= 0 H.V^T=0

Avec :

H^T La matrice transposée de la matrice H

V^T La matrice transposée de la matrice V

La matrice H s'appelle matrice de contrôle de parité de la matrice génératrice G.

$$H=[I_{n-k}, P^T] \quad (4.7)$$

$$H=\left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right] \quad (4.8)$$

Matrice identité I $[n - k]$ Matrice P^T $[k \times n-k]$

Orthogonalité entre G et H : G.H^T=0 (4.9)

4.7.Décodage

On suppose qu'au niveau de la réception on obtient un vecteur :

R$[R_1\ R_2\ R_3\ R_4\ R_5\ R_6 R_7\ R_8\ R_9\ R_{10}\ R_{11}\ R_{12}]$ $\Rightarrow$ code reçu ;

On peut écrire E= R+V=$[E_1\ E_2\ E_3\ E_4 E_5\ E_6\ E_7\ E_8\ E_9\ E_{10}\ E_{11}\ E_{12}\ ]$

Si $E_i \neq 0$ alors $R\_i \neq V\_i$ $\Rightarrow$ erreur

Si $E_i = 0$ alors $R\_i = V\_i$ $\Rightarrow$ pas d'erreur.

Le vecteur reçu :

$$R = V + E \quad (4.10)$$

4.7.1. **Syndrome**

A la réception, on calcule le vecteur syndrome S du message reçu :

S=R.H^T=$[S_1\ S_2\ S_3\ S_4]$ si :

$$S = \begin{cases} 0 & alors\ le\ mot\ codé\ est\ correct \\ 1 & \text{alors le mot codé est érroné} \end{cases}$$

$$R = V + E$$

$$S = (V + E).H^T$$

$$S = v.H^T + E.H^T$$

$$S=[E_1\ E_2\ E_3\ E_4\ E_5\ E_6\ E_7\ E_8\ E_9\ E_{10}\ E_{11}\ E_{12}]\ .\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\1&1&0&0\\1&0&1&0\\0&1&1&0\\1&1&1&0\\1&0&0&1\\0&1&0&1\\1&1&0&1\\0&0&1&1\end{bmatrix} \quad (4.11)$$

$$S=\begin{bmatrix}S_1\\S_2\\S_3\\S_4\end{bmatrix}=\begin{bmatrix}E_1+E_5+E_6+E_8+E_9+E_{11}\\E_2+E_5+E_7+E_8+E_{10}+E_{11}\\E_3+E_6+E_7+E_8+E_{12}\\E_4+E_9+E_{10}+E_{11}+E_{12}\end{bmatrix} \quad (4.12)$$

Les 8 bits du mot $r_4, r_3, r_2, r_1, C_8, C_7, C_6, C_5, C, C_3, C_2, C_1$ sont transmis en série au récepteur. Ce dernier effectue sur le mot reçu une autre opération mathématique par le calcul de quatre bits S_1, S_2, S_3, S_4 comme suit:

$S_1 = r_1 \oplus C_1 \oplus C_2 \oplus C_4 \oplus C_5 \oplus C_7$

$S_2 = r_2 \oplus C_1 \oplus C_3 \oplus C_4 \oplus C_6 \oplus C_7$

$S_3 = r_3 \oplus C_3 \oplus C_4 \oplus C_6 \oplus C_8$

$S_4 = r_4 \oplus C_4 \oplus C_5 \oplus C_6 \oplus C_7 \oplus C_8$

S_1	S_2	S_3	S_4		E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}
0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0		1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0		0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0		0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1		0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0		0	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0		0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	0		0	0	0	0	0	0	1	0	0	0	0	0
1	1	1	0		0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	1		0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	1		0	0	0	0	0	0	0	0	0	1	0	0
1	1	0	1		0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1		0	0	0	0	0	0	0	0	0	0	0	1

Tableau 4. 3: Table du décodage par syndrome

4.8. Simulation

Avant de passer à la réalisation pratique de notre système, nous avons eu recours à la simulation des différentes parties du système. Pour cela, nous avons utilisé le logiciel ISIS qui est un très bon logiciel de simulation en électronique. C'est un éditeur de schémas qui intègre un simulateur analogique, logique ou mixte. Toutes les opérations se passent dans cet environnement, aussi bien la configuration des différentes sources que le placement des sondes et le tracé des courbes. La simulation permet d'ajuster et de modifier le circuit comme si on manipulait un montage réel. Ceci permet d'accélérer le prototypage et de réduire son coût. Il faut toujours prendre en considération que les résultats obtenus de la simulation sont un peu différents de celles du monde réel, et ceci

dépend de la précision des modèles SPICE[1] des composants et de la complication des montages.

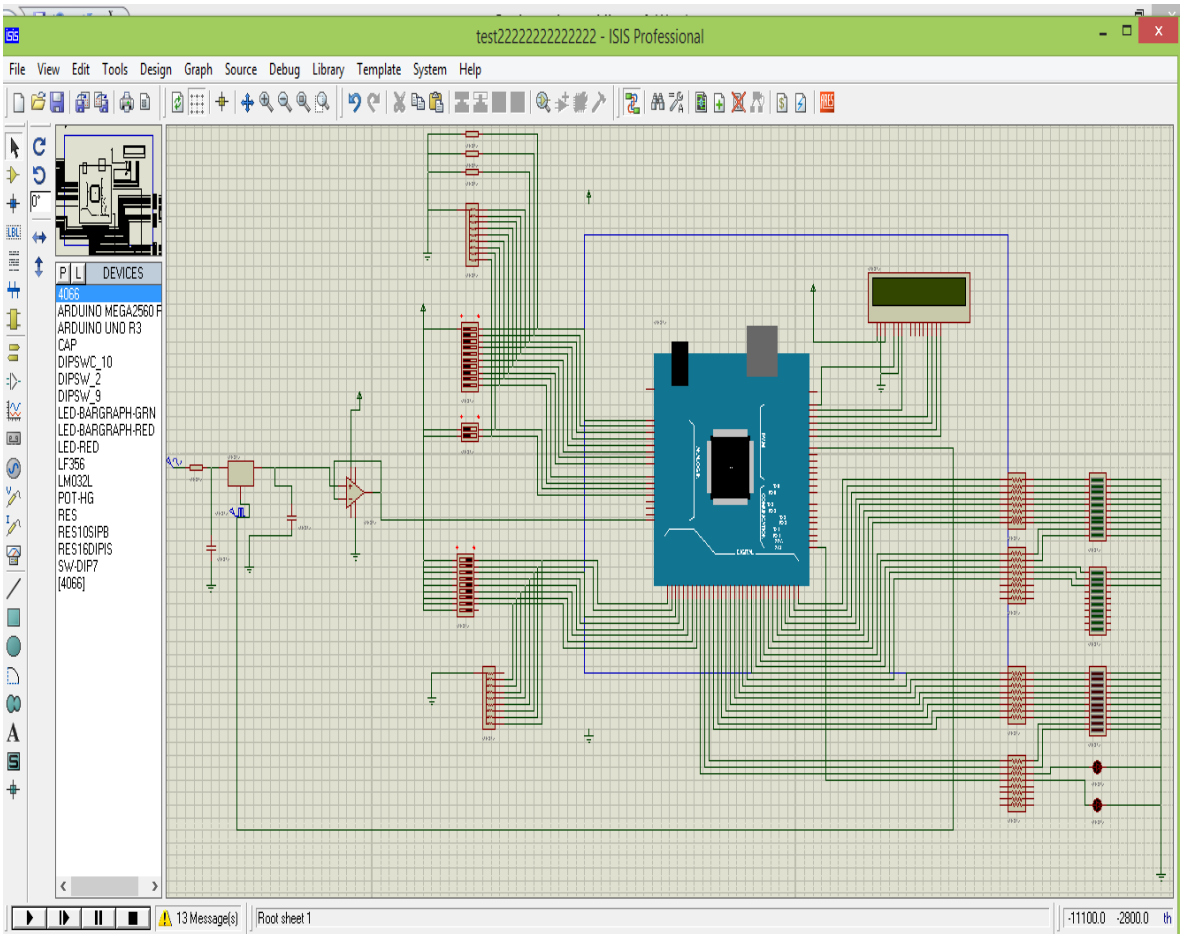


Figure 4. 16: Fenêtre du logiciel de simulation Proteus ISIS

1 **SPICE** (**S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis) est un logiciel de simulation généraliste de circuits électroniques analogiques. Il permet la simulation au niveau du composant (résistances, condensateurs, transistors) en utilisant différents types d'analyses

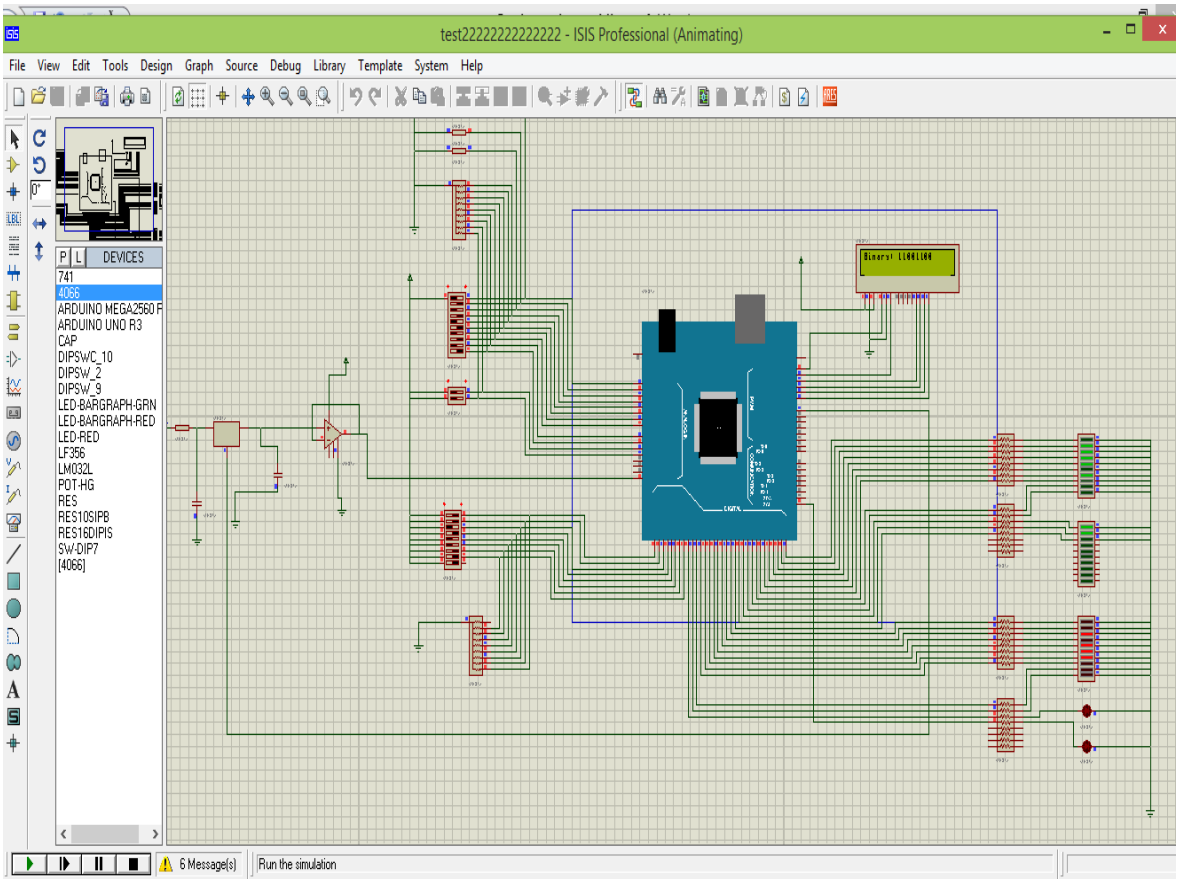


Figure 4. 17: Le logiciel ISIS en mode simulation du projet (codeur +erreur)

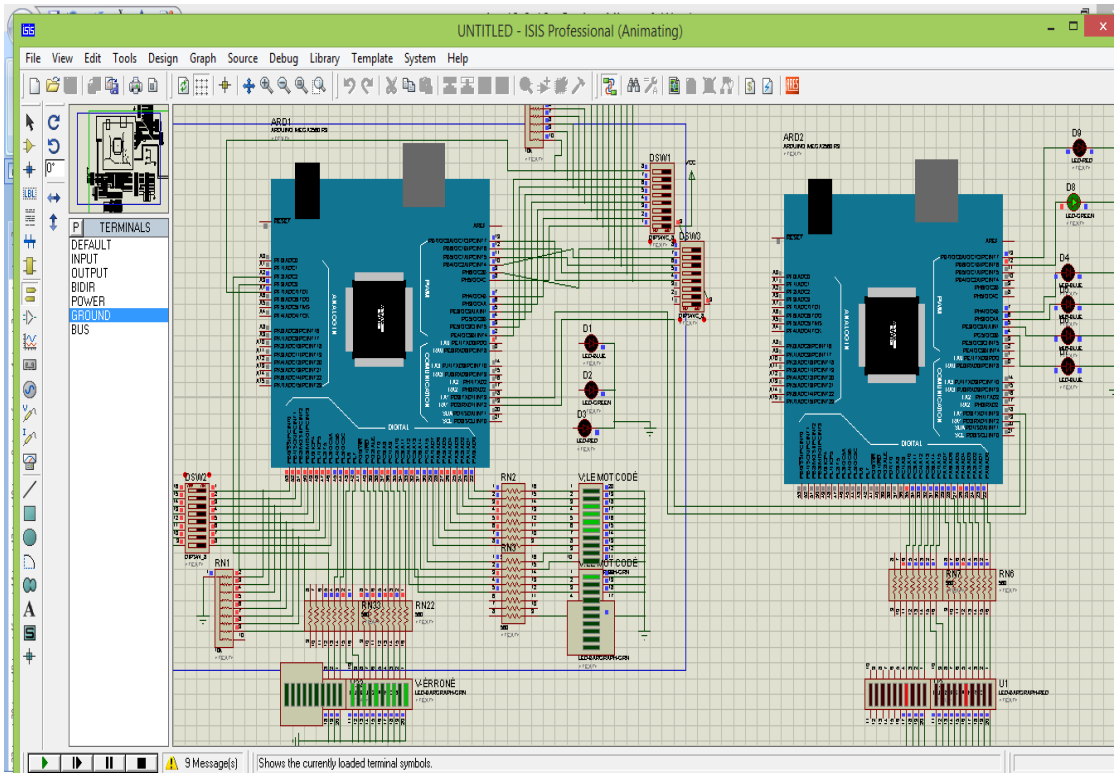


Figure 4. 18: codeurs (ARDUINO MEGA) + décodeur (ARDUINO MEGA)

4.9. Partie Logicielle

Dans notre projet, nous nous sommes intéressés à l'Arduino qui a son propre logiciel de programmation qui est opté au langage C et C++

D'une part, le langage C est utilisé dans différents systèmes et domaines de développement, ce qui nous permettra une évolution future ; d'autre part, le langage C est l'un des langages les plus puissants. Nous avons fait le choix d'utiliser l'environnement de développement IDE Arduino.

4.9.1. **Initialisation**

Pour pouvoir utiliser les différents ports de la carte Arduino, le logiciel exige qu'on lui indique comment est connecté chaque port (mode input ou output).

```
int val=A15;
int i;
int s=7;
//volatile int x=7;

// déclaration des entrées et des sorties
//les sorties
int led1=22;
int led2=23;
.
.
.

int led3=24;

//les entrées
int en1=53;
.
.
.

int en8=46;
int value1;
//types des valeurs
boolean p;
void setup() {
  // put your setup code here, to run once:
//serial.begin(9600);
//le Mode:
pinMode(led1,OUTPUT);
.
```

```
.
.
pinMode(en1,INPUT);
.
4.9.2.
pinMode(en8,INPUT);
//puis on écrit notre boucle
```

4.9.3. **Le codeur (la première carte Arduino)**

a. **Le Codage d'un mot de 8 bit**

On doit coder un mot binaire avec le code du Hamming (les calculs des ports sont déjà faits), le programme du codage sur Arduino :

C'est le même principe d'allumer une LED, à la sortie de l'Arduino on a un Baragraphe des LEDS

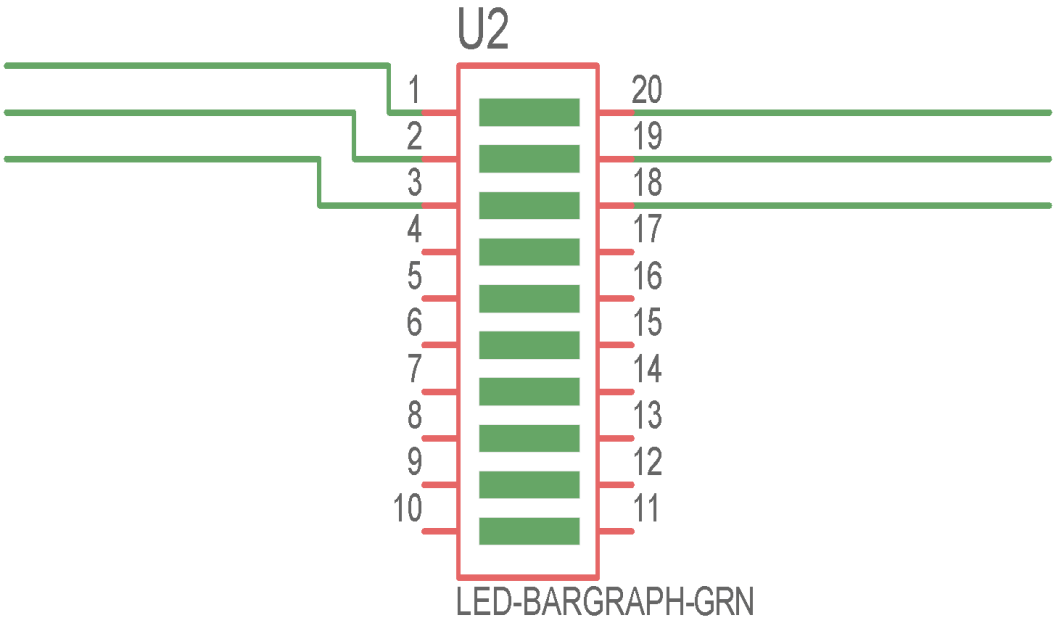


Figure 4. 19: LED-BARAGRAPH

A la fin, on obtient le mot codé par Hamming du 12 bit

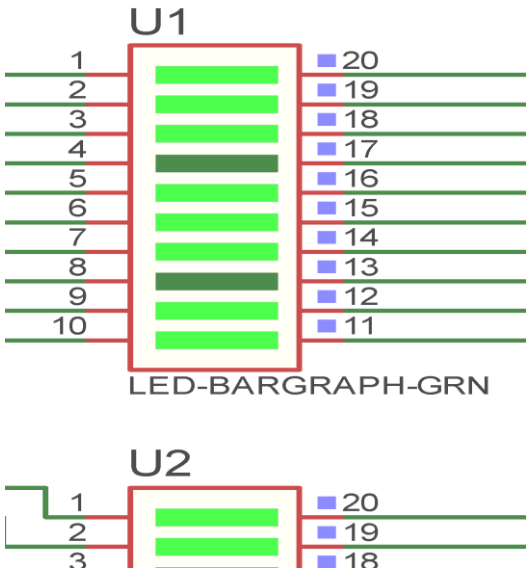


Figure 4. 20: Le mot codé du 12 bit présenté sur le Baragraph

Codage du signal analogique

1- On le convertit en un signal numérique :

Après le filtrage, on a l'échantillonnage. Dans cette, étape on va générer un signal clock à partir de la carte Arduino à l'aide des interruptions car elle ne dépend pas de la partie voide loop()

- **les interruptions**

Une interruption est une action, déclenchée par un événement précis, qui va arrêter le programme principal, s'exécuter, puis reprendre le programme principal à l'endroit où il s'était arrêté. Les interruptions peuvent être déclenchées par :

- Les Timers (pin 11 ou pin 12)
- Les pins interruptibles (PWM)
- Les interfaces de communication
- Le convertisseur analogique-numérique

Pour utiliser les interruptions, il faut tout d'abord les activer. Il faut pour cela fixer les valeurs de registre suivantes :

Exemple Timer2 :

//TCNT2=0;

//TIFR2=0x00;

//TIMSK2=0x01;

//TCCR2A=0x00;

//TCCR2B=0x05;

Il faut des interruptions pour la génération d'un signal clock qui va nous aider dans l'échantillonnage.

Ou bien, comme dans notre cas, on a généré un signal carré à partir des bronches PWM qui ont la particularité de pouvoir changer les fréquences

Programme

```
void loop() {
  // entrée le code et exécuté le avec répétition:
 // for (i=0;i<1;i+1){
  //programme générateur d'impulsion
analogWrite(s,128);
//     digitalWrite(s,HIGH);
// delay(0.0002);
// digitalWrite(s,LOW);
// delay(0.0002);
```

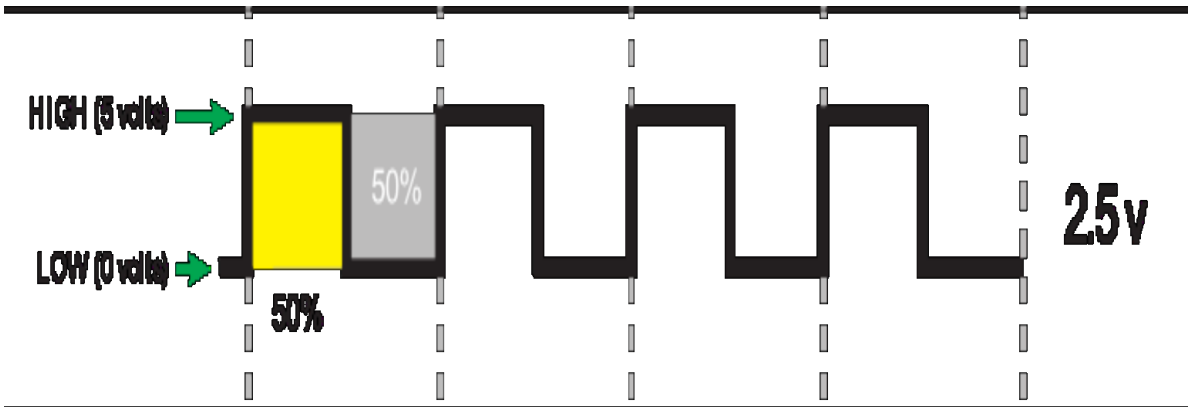


Figure 4. 21: signal de PWM à 50%

Programme de la conversion sous Arduino :

```
void loop() {

   // entrée le code et exécuté le avec répétition:

 // for (i=0;i<1;i+1){

   //

   //programme de conversion

 int valeur= analogRead(val);

  //char ch=serial.read();

int valeur1=map(valeur,0,1023,0,255);

le programme d'affichage sur LCD :

//programme du l'afficheur LCD

 lcd.print("Binary: ");

lcd.print(valeur1,BIN); delay(200);

lcd.clear();

}
```

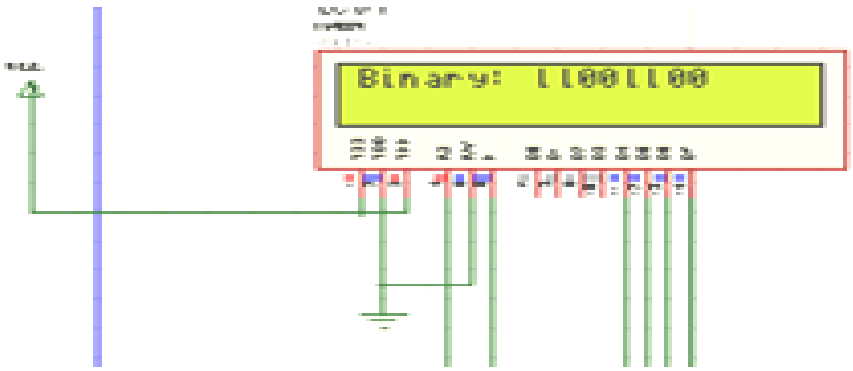


Figure 4. 22 : les résultats de la conversion sur LCD

4.9.4. **Décodeur (deuxième carte Arduino)**

a. **Décodage du vecteur reçu R**

A la réception on reçoit le vecteur R de 13 bits

R$[R_1\ R_2\ R_3\ R_4\ R_5\ R_6 R_7\ R_8\ R_9\ R_{10}\ R_{11}\ R_{12} R_{13}\ ]$

Qu'on vas le décoder pour obtenir le mot initial (émis) C

- **Détection de l'erreur**

D'abord On calcule le syndrome S de 4 bits

S=R.H^T=$[S_1\ S_2\ S_3\ S_4]$

Programme de calcul du syndrome sur Arduino 2

```
void loop() {
  //-- Si des données sont présentes
   M=Serial1.read();
   if (M >0)
   { digitalWrite(S,HIGH);}
 else
 { digitalWrite(S,LOW);}
   for (k=0;k<13;k++)//conversion M en binaire
//   {R[k]=M & (2<<k);

{digitalWrite(ledR[k],R[k]);}// allumée les leds  du mot reçu tout dépend de
l'état du R[k]
  // { digitalWrite(S1, R[k]);}//sortie mot reçu série dépand de l'état du R[k]
   delay(100);   //met la led correspondant au switch au niveau(haut ou bas) tout
dépend de l'état du switch

for (i=0;i<4;i++) /// en calcule le syndrom
{
//s[2]=R[1]^R[5]^R[7]^R[8]^R[10]^R[11];
//s[3]=R[2]^R[6]^R[7]^R[8]^R[12];
//s[4]=R[3]^R[9]^R[10]^R[11]^R[12];
   { digitalWrite(Sled[i], s[i]);}
//met la led correspondant au syndrom au niveau (haut ou bas) tout dépend de
l'état du s[i]

delay(100);
}
```

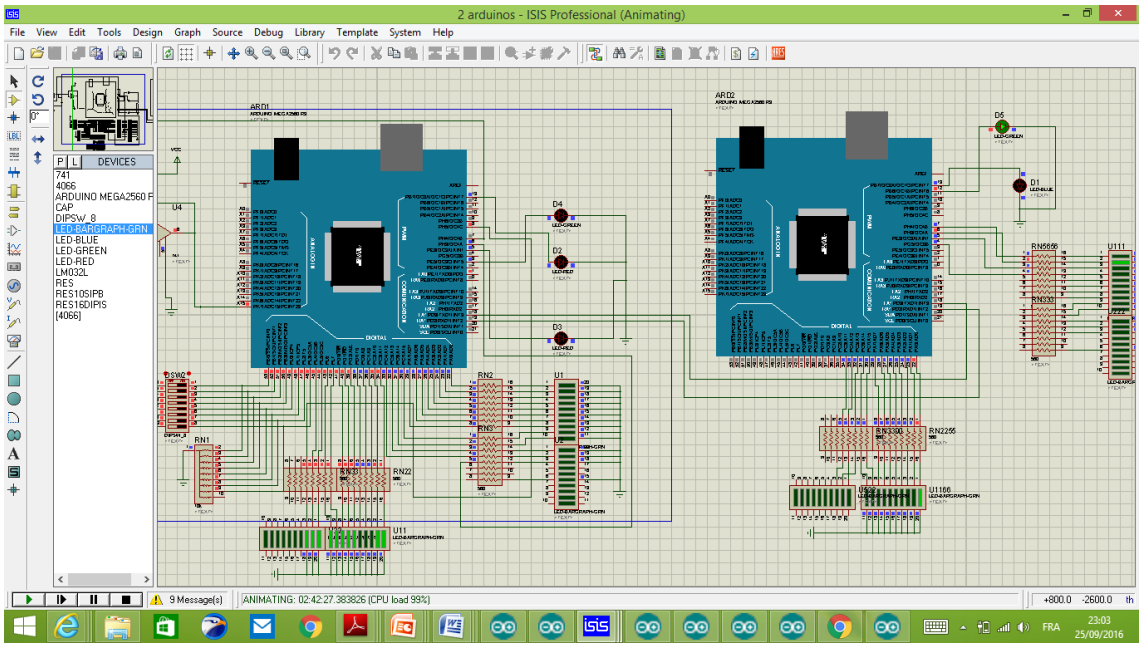


Figure 4. 23 : réception du mot et le calcul de syndrome

A partir de la valeur de S on peut détecter si on erreurs ou pas et en quel bit (voir le Tableau 4.3). Si :

S[0 0 0 0]== pas d'érreurs

- **Correction de l'erreur**

Le vecteur reçu :

$$R = V + E$$

Alors :

$$V = R \oplus E$$

Donc on fait entrée un vecteur E de 13 bits

$$\mathbf{R}[E_1\ E_2\ E_3\ E_4\ E_5\ E_6 E_7\ E_8\ E_9\ E_{10}\ E_{11}\ E_{12} E_{13}\ ]$$

Programme de correction de l'erreur

```
///correction de l'erreur
int ledE[13]={pin de l'entrée};//déclaration des pattes des leds  correspond à E

int SwE[13]={4,5,6,7,8,9,10,11,12,A7,A6,A5,A4};//déclaration des switch erreurs
boolean valueE[13]; //état de switch erreur
boolean E[13];//vecteur erronné
///int compteur;
boolean V[13]; // Déclaration des bits du mot

void setup() {  //fonction permettant d'intialiser le matériel (leds et switch
 //-- définition du port Série Logiciel
Serial.begin(9600); // indique que Arduino se situe sur le port USB 9600
///pinMode(leds,OUTPUT);//mot codé série
for (i=0;i<13; i++){   //boucle parcourant toutes les variable de 0 à 12
pinMode(SwE[i],INPUT);  // indique que le switch est une entrée
```

```
for (l=0;l<13;l++){
  pinMode(ledE[l],OUTPUT);// leds correspond à mot codé erroné
  valueE[k]=digitalRead(SwE[k]);

V [k]= E[k]^Rk];
//if (V [k]==1)
//{digitalWrite(ledE[k],HIGH);}
//else
//{digitalWrite(ledE[k],LOW);}
```

Depuis le nouveau vecteur V calculé on peut distingue le mot initial C

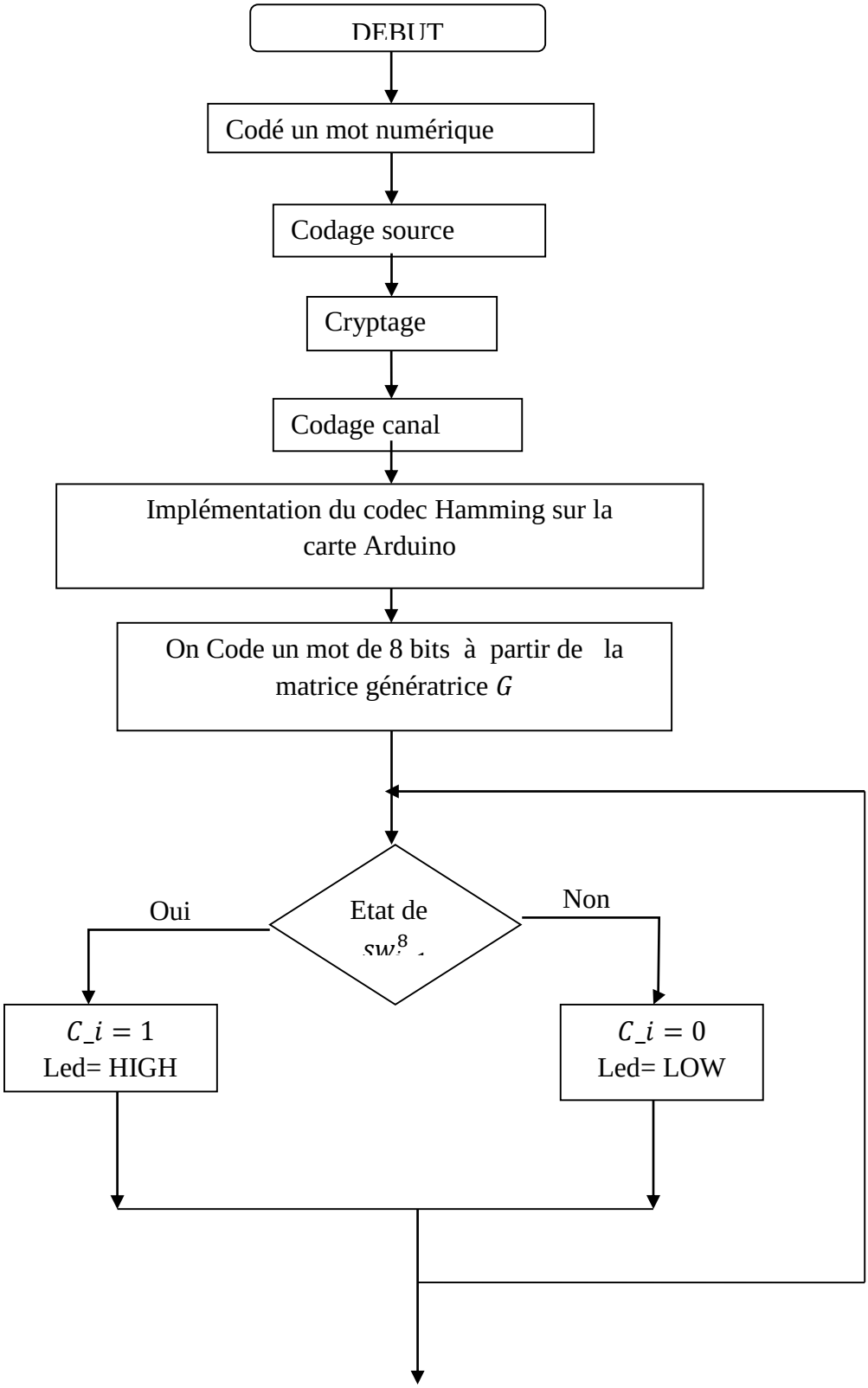
Alors :

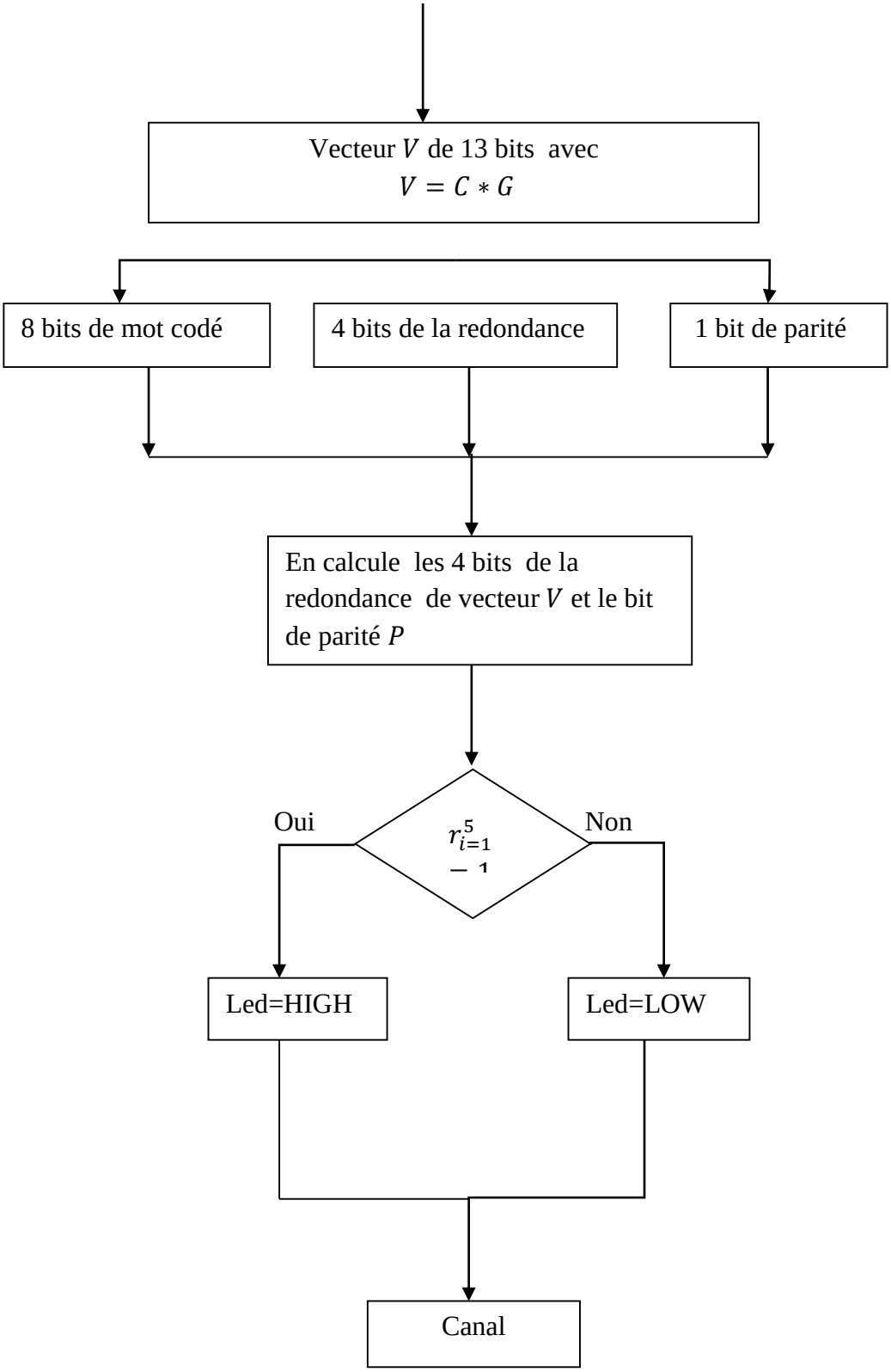
$\boldsymbol{C = [\, v_6\, , v_7, v_8, v_9, v_{10}, v_{11}, v_{12} v_{13}]}$

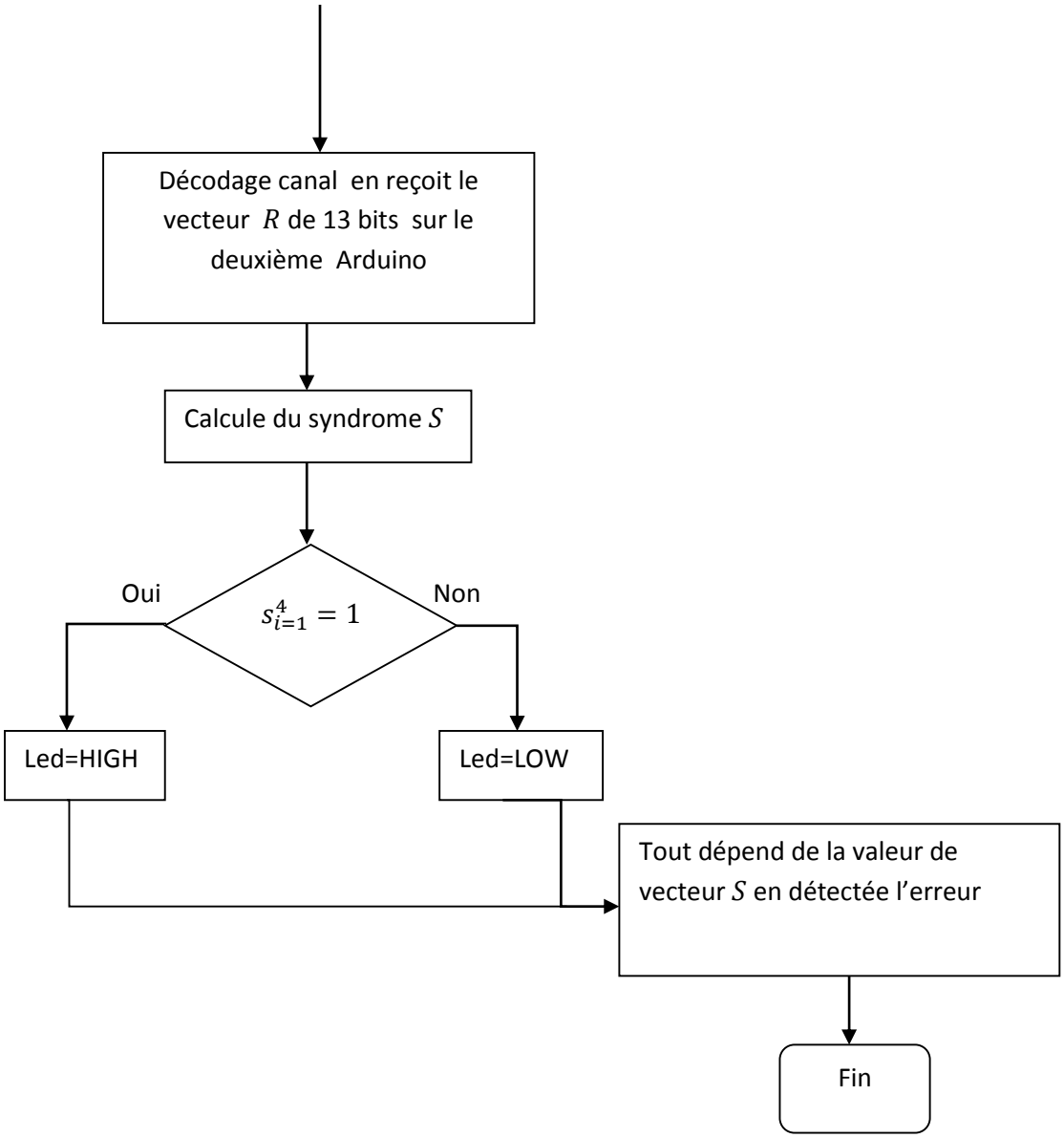
Programme sur deuxième carte Arduino

```
int ledC[8]={choisir les pins}leds du mot décoder
Void loop{
for (k=5 ;k<13 ;k++) {// boucle parcourant les variable de 5 à 12
{for (i=0 ;i<8 ;i++){
C[i]=V[k] ;
//if (C [i]==1)
//{digitalWrite(ledC[i],HIGH);}
//else
//{digitalWrite(ledC[i],LOW);}
```

Organigramme




Vecteur V de 13 bits avec
$V = C * G$
8 bits de mot codé
4 bits de la redondance
1 bit de parité
En calcule les 4 bits de la redondance de vecteur V et le bit de parité P
Oui
$r_{i=1}^{5}$
Non
Led=HIGH
Led=LOW
Canal


Décodage canal en reçoit le vecteur R de 13 bits sur le deuxième Arduino
Calcule du syndrome S
Oui
$s_{i=1}^{4} = 1$
Non
Led=HIGH
Led=LOW
Tout dépend de la valeur de vecteur S en détectée l'erreur
Fin

4.10. Conclusion

La réalisation de ce projet nous a énormément appris tant au niveau de l'électronique que de l'implémentation du code Hamming sur Arduino. Nous avons aussi acquis de nouvelles connaissances au niveau de la gestion du temps et des équipes.

Ce travail reste, comme toute œuvre humaine, incomplet et perfectible. Nous recommandons d'en améliorer la conception et pour cela, nous proposons ci-dessous des améliorations pour les futurs développements :

Améliorer la protection des entrées analogiques.

Améliorer la protection du canal contre les perturbations externes et même internes

Les améliorations qu'on peut apporter à notre travail sont énormes et peuvent varier selon le type d'application qu'on souhaite.

Conclusion générale

Nous avons donné un aperçu du codage canal. Qu'est utilisé pour corriger les erreurs qui peuvent apparaître lors d'une transmission sur un canal. Ces erreurs peuvent provenir d'un mauvais réglage des équipements ou de leur déréglage. Ainsi, il permet de s'affranchir de certaines imperfections technologiques et matérielles.

Les codes en bloc linéaires, quant à eux, sont une classe très répandue car ils offrent des facilités de manipulations tant pratiques que théoriques.

Les codes correcteurs d'erreurs sont un outil visant à améliorer la fiabilité des transmissions sur un canal bruité. La méthode qu'ils utilisent consiste à envoyer sur le canal plus de données que la quantité d'informations à transmettre. Une redondance est ainsi introduite. Si cette redondance est structurée de manière exploitable, il est alors possible de corriger d'éventuelles erreurs introduites par le canal. On peut alors, malgré le bruit, retrouver l'intégralité des informations transmises au départ.

Nous y avons montré quels pouvaient être les critères pour construire un bon code ainsi que la technique optimale de décodage et une de ses implémentations via le décodage par syndrome. Néanmoins, le domaine du codage est, bien évidemment, beaucoup plus vaste.

Nous avons utilisé deux cartes Arduino Mega pour appliquer le code de Hamming en vue de la détection et la correction d'erreurs sur un mot numérique et un signal analogique transmis séparément, une comme un codeur et l'autre comme un décodeur.

Pour la conversion d'un signal analogique en numérique A/N, on a appliqué la modulation MIC sur notre signal par un filtre passe bas et un échantillonneur-bloqueur, puis nous avons utilisé une carte Arduino pour convertir ce signal.

Le codage du mot et le calcul de sa redondance ont été effectués au moyen de la matrice génératrice G.

Le décodage a été fait par la matrice de contrôle de parité H.

Dans le code de Hamming, chaque bit de contrôle est pris comme un mod 2 somme d'un sous-ensemble des bits de message.

Compte tenu de la configuration R reçue, le décodeur doit éventuellement décider si le mot de code transmis est erroné ou non. Si le décodeur est capable de trouver le motif d'erreur e, alors le mot de code sera V= R+ e.

Le décodage devient ensuite très simple puisqu'il suffira de garder le bit majoritaire dans chaque bloc.

La matrice de contrôle de parité d'un code nous fournit un moyen de détermination d'une borne inférieure sur la distance minimale du code, sans avoir à examiner toutes les paires de mots de code ni de voir les syndromes.

Dans notre travail, nous avons appliqué une erreur afin de la détecter au moyen d'un syndrome, ce qui nous permet de vérifier le bon fonctionnement de ce dernier. Le Code de Hamming peut détecter jusqu'à trois erreurs mais ne peut en corriger qu'une seule.

L'objectif de notre travail a été non seulement de concevoir des architectures de codeur/décodeur rapides mais également de les sécuriser, autrement dit, de les rendre sûres de fonctionnement

Annexes

Annexes

Pour différentes combinaisons du vecteur U, on obtient les différentes combinaisons du vecteur V

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	r_1	r_2	r_3	r_4	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0	0	1
0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	0	1	1
0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	1	1	0	1
0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	1	1	0
0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0
0	0	0	1	0	0	1	1	0	0	1	0	0	0	0	1	0	0	1	1
0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0
0	0	0	1	0	1	0	1	1	0	1	0	0	0	0	1	0	1	0	1
0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	1	0	1	1	0
0	0	0	1	0	1	1	1	0	1	0	1	0	0	0	1	0	1	1	1
0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	1
0	0	0	1	1	0	1	0	1	0	1	0	0	0	0	1	1	0	1	0
0	0	0	1	1	0	1	1	1	0	0	1	0	0	0	1	1	0	1	1
0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	1	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	1	1	1	0	0	1	1	1	0	0	0	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1
0	0	1	0	0	0	1	0	1	0	1	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	1
0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0

0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1
0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0
0	0	1	0	0	1	1	1	1	1	0	1	0	0	1	0	0	1	1	1
0	0	1	0	1	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1
0	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	1	0	1	0
0	0	1	0	1	0	1	1	0	0	0	1	0	0	1	0	1	0	1	1
0	0	1	0	1	1	0	0	1	0	1	0	0	0	1	0	1	1	0	0
0	0	1	0	1	1	0	1	1	0	0	1	0	0	1	0	1	1	0	1
0	0	1	0	1	1	1	0	0	1	1	1	0	0	1	0	1	1	1	0
0	0	1	0	1	1	1	1	0	1	1	0	0	0	1	0	1	1	1	1
0	0	1	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	1	1	0	1	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	1	1	0	1	0	0	1	1	0	1	0	0
0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	1	0	1	0	1
0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	0	1	1	0
0	0	1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	1	1	1
0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	1	0	0	1	0	0	0	1	1	1	0	0	1
0	0	1	1	1	0	1	0	1	1	0	0	0	0	1	1	1	0	1	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0	1	1	1	0	1	1
0	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	1	0	1	1	1	0	0	1	1	1	1	0	1
0	0	1	1	1	1	1	0	1	0	0	1	0	0	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0	1	1	1	0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1
0	1	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	1	0	1
0	1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	1	1	0
0	1	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	1	1	1
0	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0
0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
0	1	0	0	1	0	1	0	1	1	1	0	0	1	0	0	1	0	1	0
0	1	0	0	1	0	1	1	1	1	0	1	0	1	0	0	1	0	1	1
0	1	0	0	1	1	0	0	0	1	1	0	0	1	0	0	1	1	0	0
0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	0	1	1	0	1
0	1	0	0	1	1	1	0	1	0	1	1	0	1	0	0	1	1	1	0
0	1	0	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1
0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0
0	1	0	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	1	1	0	1	0	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	0

0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	1	1	0	0	0	1	0	1	0	1	1	0
0	1	0	1	0	1	1	1	1	1	1	1	0	1	0	1	0	1	1	1
0	1	0	1	1	0	0	0	1	1	0	1	0	1	0	1	1	0	0	0
0	1	0	1	1	0	0	1	1	1	1	0	0	1	0	1	1	0	0	1
0	1	0	1	1	0	1	0	0	0	0	0	0	1	0	1	1	0	1	0
0	1	0	1	1	0	1	1	0	0	1	1	0	1	0	1	1	0	1	1
0	1	0	1	1	1	0	0	1	0	0	0	0	1	0	1	1	1	0	0
0	1	0	1	1	1	0	1	1	0	1	1	0	1	0	1	1	1	0	1
0	1	0	1	1	1	1	0	0	1	0	1	0	1	0	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1
0	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	1	1	1	0	1	1	0	0	0	0	1
0	1	1	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0	1	0
0	1	1	0	0	0	1	1	0	0	1	0	0	1	1	0	0	0	1	1
0	1	1	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0
0	1	1	0	0	1	0	1	1	0	1	0	0	1	1	0	0	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	1
0	1	1	0	1	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0
0	1	1	0	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
0	1	1	0	1	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0
0	1	1	0	1	0	1	1	1	0	1	1	0	1	1	0	1	0	1	1
0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	0	1	1	0	0
0	1	1	0	1	1	0	1	0	0	1	1	0	1	1	0	1	1	0	1
0	1	1	0	1	1	1	0	1	1	0	0	0	1	1	0	1	1	1	0
0	1	1	0	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	1
0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	0	1	1	1	1	0	1	1	1	0	0	1	0
0	1	1	1	0	0	1	1	1	1	0	0	0	1	1	1	0	0	1	1
0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0
0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	1	0	1
0	1	1	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1	1	0
0	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0	0
0	1	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1
0	1	1	1	1	0	1	0	0	1	1	1	0	1	1	1	1	0	1	0
0	1	1	1	1	0	1	1	0	1	0	1	0	1	1	1	1	0	1	1
0	1	1	1	1	1	0	0	1	1	1	0	0	1	1	1	1	1	0	0
0	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1
0	1	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	1
1	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	0	0

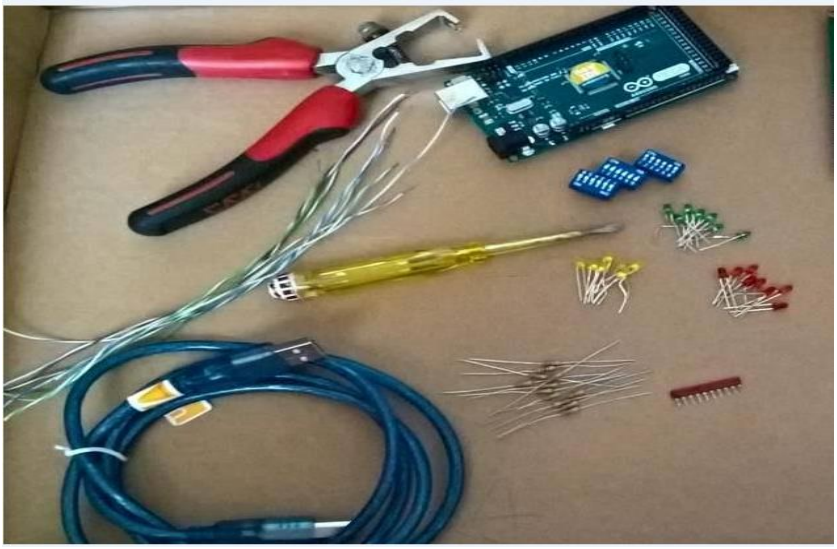
1	0	0	0	0	1	0	1	1	0	1	0	1	0	0	0	0	1	0	1
1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	1	0
1	0	0	0	0	1	1	1	0	1	1	1	1	0	0	0	0	1	1	1
1	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0	1	0	0	1
1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	1	0
1	0	0	0	1	0	1	1	1	0	1	1	1	0	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	0
1	0	0	0	1	1	0	1	0	0	1	1	1	0	0	0	1	1	0	1
1	0	0	0	1	1	1	0	1	1	0	1	1	0	0	0	1	1	1	0
1	0	0	0	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1
1	0	0	1	0	0	0	0	1	0	1	1	1	0	0	1	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	0	1	0	1	1	1	1	1	0	0	1	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1	0	0	1	1
1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	1	0	1	0	0
1	0	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	1	0	1
1	0	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1	1	0
1	0	0	1	0	1	1	1	1	0	0	1	1	0	0	1	0	1	1	1
1	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0	0	0	1	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1	1
1	0	0	1	1	1	0	0	1	1	1	0	1	0	0	1	1	1	0	0
1	0	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	1	0	1
1	0	0	1	1	1	1	0	0	0	1	1	1	0	0	1	1	1	1	0
1	0	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0
1	0	1	0	0	0	0	1	1	0	1	1	1	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0
1	0	1	0	0	0	1	1	0	1	0	0	1	0	1	0	0	0	1	1
1	0	1	0	0	1	0	0	1	0	1	1	1	0	1	0	0	1	0	0
1	0	1	0	0	1	0	1	1	1	0	0	1	0	1	0	0	1	0	1
1	0	1	0	0	1	1	0	0	0	1	0	1	0	1	0	0	1	1	0
1	0	1	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1	1	1
1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0
1	0	1	0	1	0	0	1	0	0	0	0	1	0	1	0	1	0	0	1
1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	1
1	0	1	0	1	1	0	0	0	1	1	0	1	0	1	0	1	1	0	0
1	0	0	0	1	1	0	1	0	1	0	1	1	0	1	0	1	1	0	1
1	0	0	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0
1	0	0	0	1	1	1	1	1	0	0	0	1	0	1	0	1	1	1	1
1	0	1	1	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0
1	0	1	1	0	0	0	1	0	1	1	1	1	0	1	1	0	0	0	1
1	0	1	1	0	0	1	0	1	0	0	1	1	0	1	1	0	0	1	0
1	0	1	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0	1	1
1	0	1	1	0	1	0	0	0	0	0	1	1	0	1	1	0	1	0	0

1	0	1	1	0	1	0	1	0	0	1	0	1	0	1	1	0	1	0	1
1	0	1	1	0	1	1	0	1	0	0	0	1	0	1	1	0	1	1	0
1	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1
1	0	1	1	1	0	0	0	0	1	0	0	1	0	1	1	1	0	0	0
1	0	1	1	1	0	0	1	1	1	1	0	1	0	1	1	1	0	0	1
1	0	1	1	1	0	1	0	0	0	0	0	1	0	1	1	1	0	1	0
1	0	1	1	1	0	1	1	0	0	1	1	1	0	1	1	1	0	1	1
1	0	1	1	1	1	0	0	1	0	0	1	1	0	1	1	1	1	0	0
1	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	1	0	1
1	0	1	1	1	1	1	0	0	1	0	1	1	0	1	1	1	1	1	0
1	0	1	1	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	1
1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1
1	1	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	0	1	0
1	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0
1	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	1
1	1	0	0	0	1	1	0	1	1	1	0	1	1	0	0	0	1	1	0
1	1	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0	0	1	1	0	1	1	0	0	1	0	0	0
1	1	0	0	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1
1	1	0	0	1	0	1	0	0	1	1	0	1	1	0	0	1	0	1	0
1	1	0	0	1	0	1	1	0	0	0	1	1	1	0	0	1	0	1	1
1	1	0	0	1	1	0	0	1	0	1	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1
1	1	0	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	0
1	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1	1	1	1
1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0
1	1	0	1	0	0	0	1	1	0	1	1	1	1	0	1	0	0	0	1
1	1	0	1	0	0	1	0	0	1	0	1	1	1	0	1	0	0	1	0
1	1	0	1	0	0	1	1	0	1	1	0	1	1	0	1	0	0	1	1
1	1	0	1	0	1	0	0	1	1	0	1	1	1	0	1	0	1	0	0
1	1	0	1	0	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1
1	1	0	1	0	1	1	0	0	0	0	1	1	1	0	1	0	1	1	0
1	1	0	1	0	1	1	1	0	0	1	1	1	1	0	1	0	1	1	1
1	1	0	1	1	0	0	0	1	0	1	0	1	1	0	1	1	0	0	0
1	1	0	1	1	0	0	1	0	0	1	0	1	1	0	1	1	0	0	1
1	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1	1	0	1	0
1	1	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	1	1
1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	1	1	1	0	0
1	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	0	1
1	1	0	1	1	1	1	0	1	0	0	1	1	1	0	1	1	1	1	0
1	1	0	1	1	1	1	1	1	0	1	0	1	1	0	1	1	1	1	1
1	1	1	0	0	0	0	0	1	0	1	1	1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	1	0	0	1	1	1	1	1	0	0	0	0	1
1	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1
1	1	1	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0

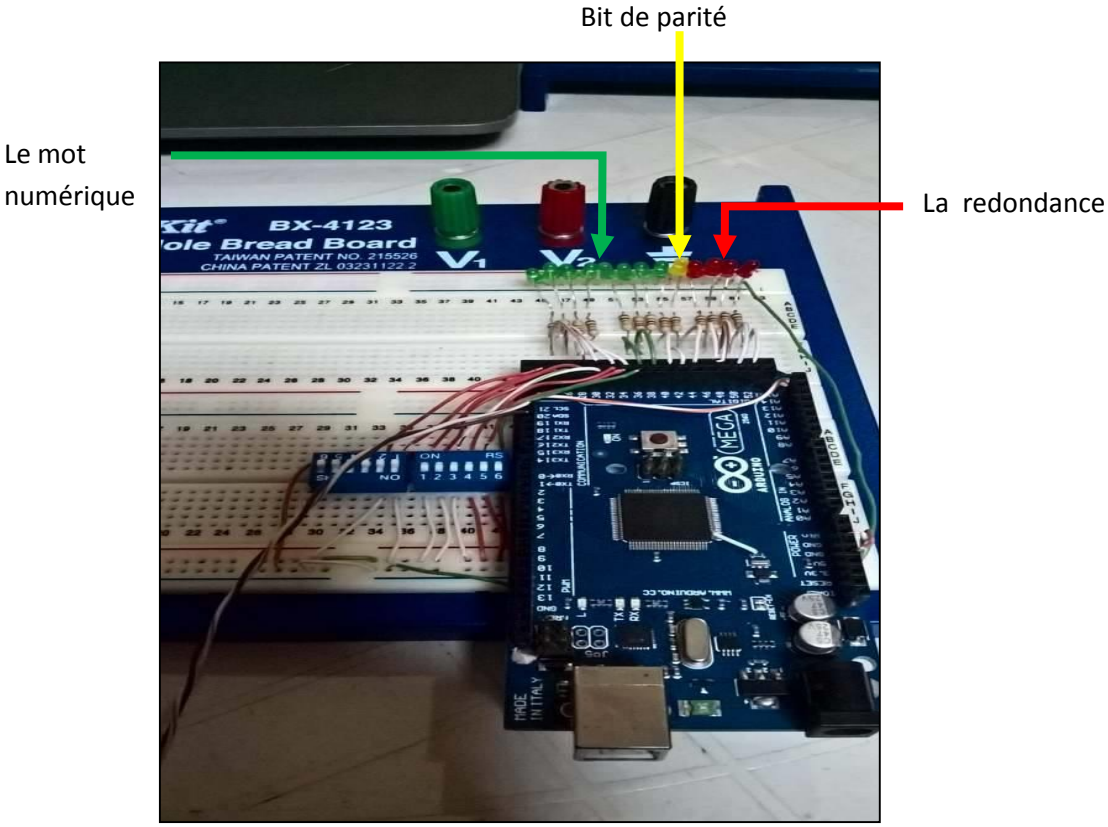
1	1	1	0	0	0	0	1	**0**	**1**	**1**	**0**	1	1	1	0	0	0	0	1
1	1	1	0	0	0	1	0	**1**	**0**	**0**	**0**	1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	1	**1**	**0**	**1**	**1**	1	1	1	0	0	0	1	1
1	1	1	0	1	0	0	0	**0**	**0**	**0**	**1**	1	1	1	0	1	0	0	0
1	1	1	0	1	0	0	1	**1**	**0**	**1**	**0**	1	1	1	0	1	0	0	1
1	1	1	0	1	0	1	0	**0**	**1**	**0**	**0**	1	1	1	0	1	0	1	0
1	1	1	0	1	0	1	1	**0**	**1**	**1**	**1**	1	1	1	0	1	0	1	1
1	1	1	0	1	1	0	0	**1**	**1**	**0**	**0**	1	1	1	0	1	1	0	0
1	1	1	0	1	1	0	1	**1**	**1**	**1**	**1**	1	1	1	0	1	1	0	1
1	1	1	0	1	1	1	0	**0**	**0**	**0**	**1**	1	1	1	0	1	1	1	0
1	1	1	0	1	1	1	1	**0**	**0**	**1**	**0**	1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0	**0**	**1**	**1**	**1**	1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	1	**1**	**1**	**0**	**1**	1	1	1	1	0	0	0	1
1	1	1	1	0	0	1	0	**0**	**0**	**1**	**1**	1	1	1	1	0	0	1	0
1	1	1	1	0	0	1	1	**0**	**0**	**0**	**0**	1	1	1	1	0	0	1	1
1	1	1	1	0	1	0	0	**1**	**0**	**1**	**1**	1	1	1	1	0	1	0	0
1	1	1	1	0	1	0	1	**1**	**0**	**0**	**0**	1	1	1	1	0	1	0	1
1	1	1	1	0	1	1	0	**0**	**1**	**1**	**0**	1	1	1	1	0	1	1	0
1	1	1	1	0	1	1	1	**0**	**1**	**0**	**1**	1	1	1	1	0	1	1	1
1	1	1	1	1	0	0	0	**0**	**1**	**1**	**1**	1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	1	**0**	**1**	**0**	**0**	1	1	1	1	1	0	0	1
1	1	1	1	1	0	1	0	**1**	**0**	**1**	**0**	1	1	1	1	1	0	1	0
1	1	1	1	1	0	1	1	**1**	**0**	**0**	**1**	1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	0	**0**	**0**	**1**	**1**	1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1	**0**	**0**	**0**	**0**	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	0	**1**	**1**	**1**	**1**	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	**1**	**1**	**0**	**0**	1	1	1	1	1	1	1	1

- Les pièces utilisent

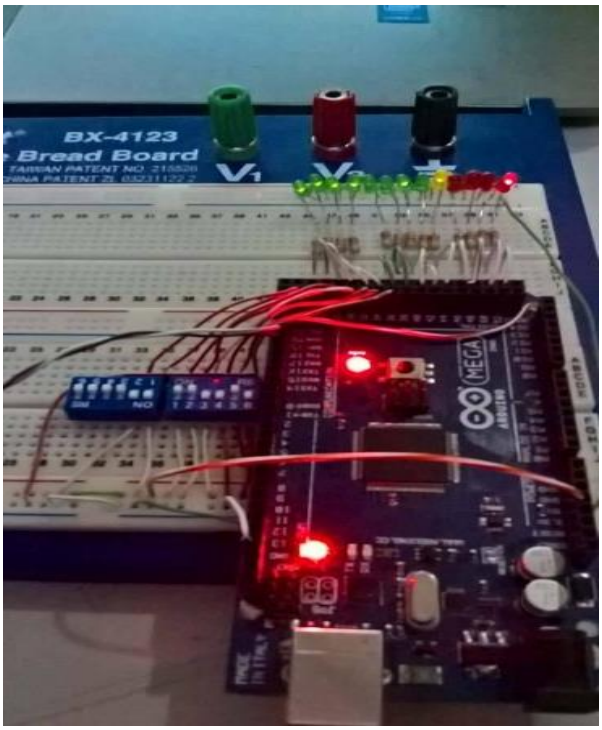
- Deux cartes Arduino Mega 2560
- Les switches
- Les LEDs : rouge, vert, jaune
- Les résistances à 10 k
- Les résistances de mise à niveaux
- Les fils de capelage
- Deux Câble USB



- Réalisation du code de Hamming sur la carte Arduino :



- Le mot codé : simulation + réalisation sur la carte Arduino



- Le mot codé + le bit de parité(en jaune) allumée:

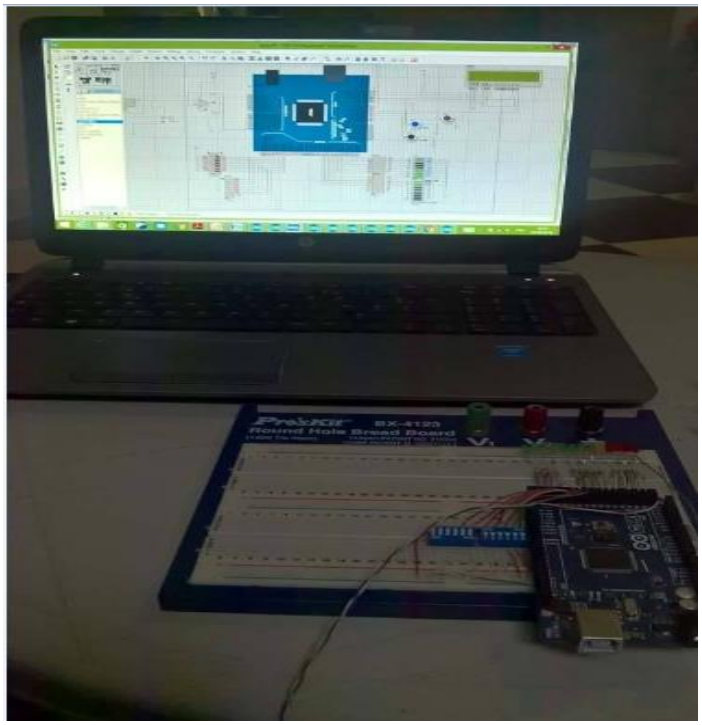
Exemple1 : mot numérique de 8 bits

C=[$C_1C_2C_3C_4C_5C_6C_7C_8$]

C = [11110010]

$V = C * G$

V= [11110010**10001**]



- Le mot codé + le bit de parité éteindre:

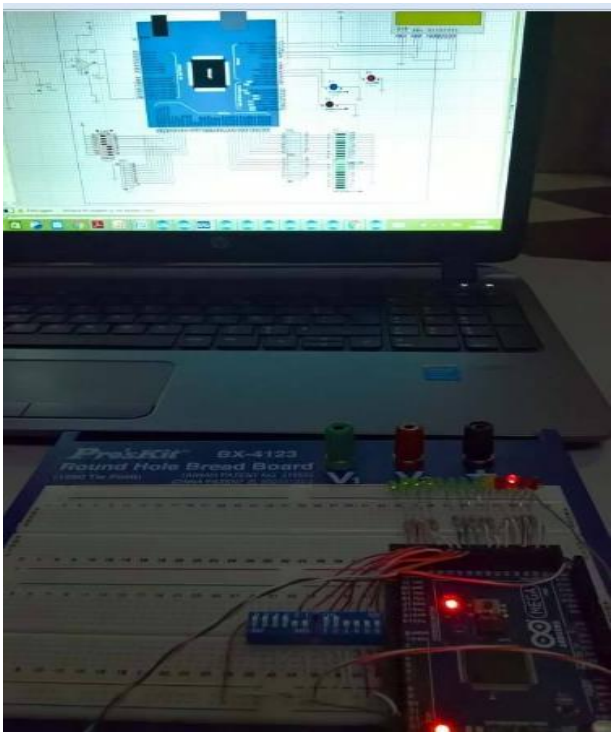
Exemple1 : mot numérique de 8 bits

C=[$C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8$]

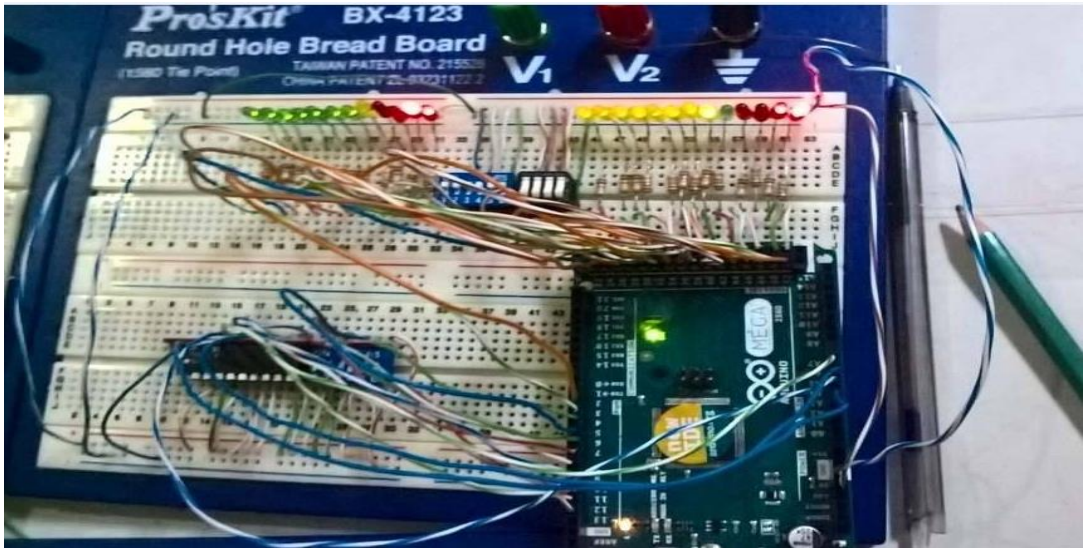
$C = [11110000]$

$V = C * G$

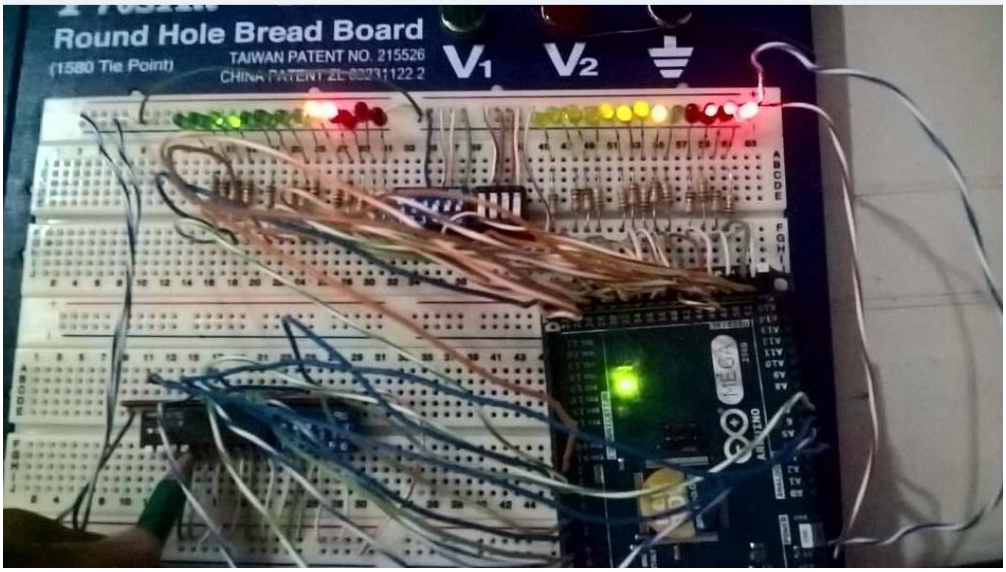
V= [11110000**0100**]



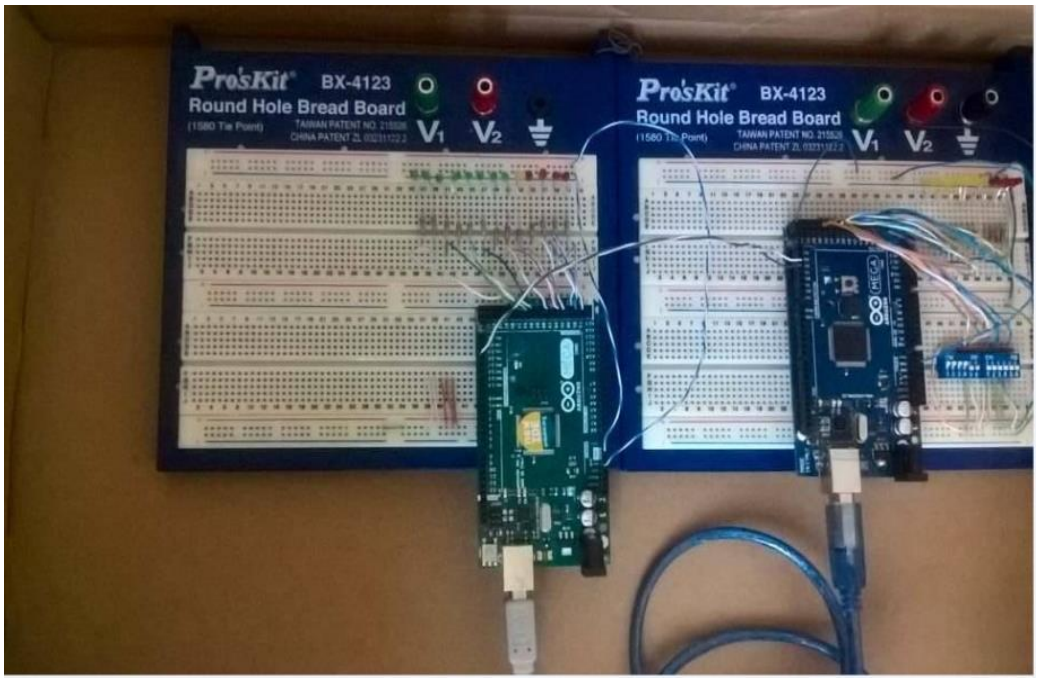
- le mot codé sans erreur



- on a 11 bits erronés



- Émission réception



Bibliographie

[1] Codage canal et turbo-codes 15/09/2000.

[2] BERROU C., PYNDIAH R., JEZEQUEL M., GLAVIEUX A., ADDE P., « La double correction des Turbo-Codes », La Recherche, n°315, pp. 34-37, décembre 1998.

[3] Jean Dion. Etude et implémentation d'une architecture de décodage générique et flexible pour codes correcteurs d'erreurs avances. Théorie de l'information et codage [math.IT]. Télécom Bretagne, Université de Bretagne Occidentale, 2013. Français.

[4] Assia mounir. Anas Bennani. Anouar. Loukili.Ali Broma sidibe. Les codes convolutifs

[5]. A.migan.S. argentieri transmission de l'information les codes convolutifs 2011/2012 .page

[6] BERROU C., « Les turbo codes convolutifs », ENST Bretagne, mars 1999.page(49)

[7] Mr. Chabane. Mlle Allali.N.turbo code. Page (21, 22)

[8] Hikmet SARI . Transmission des signaux numériques page (20, 21, 22)

[9] Jean-Claude Belfiore, Philippe Ciblat, Michèle WiggerCommunications Numériques

Et Théorie de l'Information

[10] jean-Yves –chouinard. GEL-7064 : théorie et pratique des codes correcteurs code linéaire Notes de code, 4mars 2013.

[11] S.Foughali, S.Khelifa, « Concaténation des Codes Cyclique (Reed Solomon –

Hamming) Appliquées aux images fixes », Institut d'Informatique, USTO 1998 .

[12] Kossigan Roland ASSILEVI (Togo), Mamadou COULIBALY (Mali), Maurin DONNEAUD (France), Cédric DOUT RIAUX (France), Simon LAROCHE (Canada-

Québec), Florian PIT T ET (Suisse), Marie-Paul UWASE (Rwanda), « *LIVRE ARDUINO* »,2011.

[13] initialisation à Arduino .

[14] Khaddi yousef. Oubarka othman.oucharra houssaine.oulddaouuia abdelali. SYSTÈME DE CONTRÔLE ET DE COMMANDE D'ACCÈS À DISTANCE VIA GSM.IGA 2013/2014.

[15] https://www.arduino.cc/en/Main/ArduinoBoardMega2560

[16] *www.mon-club-elec.fr/pmwiki_**reference_arduino**/pmwiki.php*

[17] http://www.embarcados.com.br/arduino-mega-2560/

[18] Louis REYNIER, « C'est quoi Arduino ? »,2011.

[19] http://wiki.mchobby.be/index.php?title=3D-OrdBot-Arduino

[20] Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V