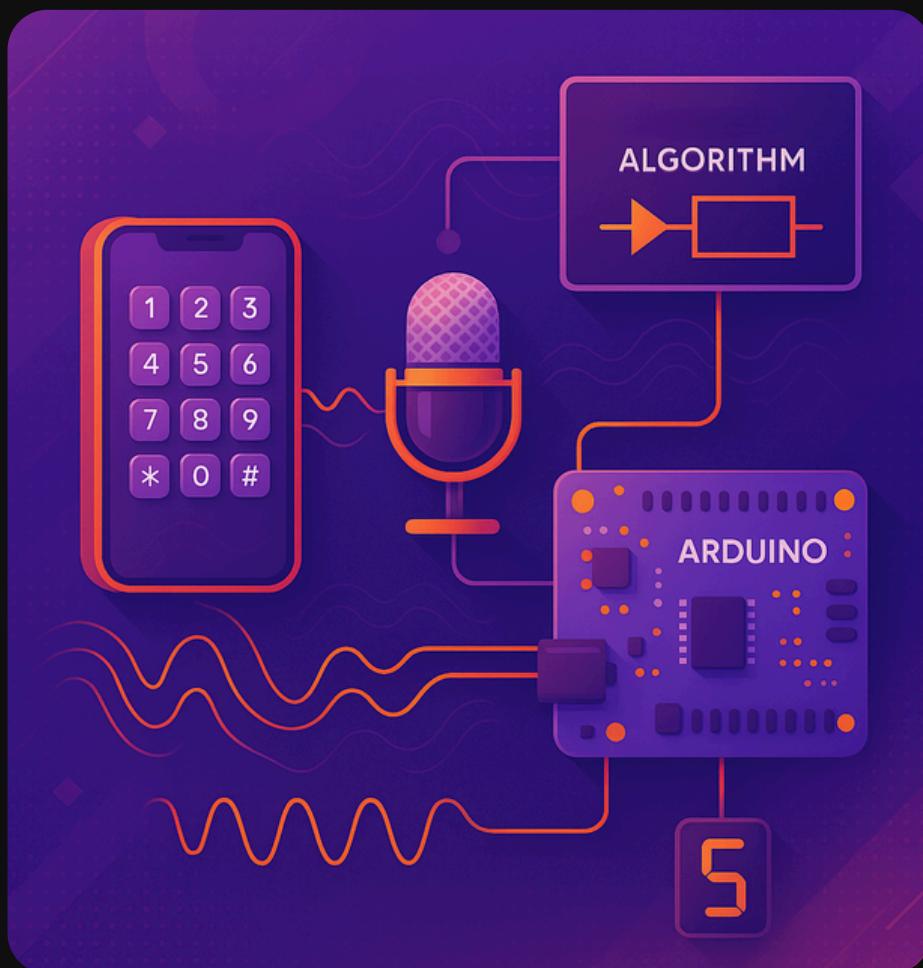


RAPPORT PROJET

Implémentation d'un système pour décoder les signaux DTMF et allumer des LED correspondantes



**ENSA TÉTOUAN
2024-2025**

Préparé par :
BOUARRAF Doha
DIALLO Abdoul-Moumouni
MAKRI Youssra
SIMPORE Taobata

Professeur :
Zakaria Errachidi

Sommaire

→ **01** Introduction

→ **02** Théorie

→ **03** Matériel et outils

→ **04** Implémentation

→ **05** Résultats

→ **06** Conclusion

→ **07** Annexes

Introduction



Les signaux DTMF (Dual Tone Multi-Frequency), ou multifréquences à double tonalité, sont des signaux utilisés dans les systèmes de téléphonie pour transmettre les chiffres composés via le clavier. Chaque touche appuyée génère une combinaison unique de deux fréquences, l'une basse et l'autre haute. Ce système est utilisé dans les appels téléphoniques, les serveurs vocaux interactifs (SVI/IVR), ainsi que dans des applications de contrôle à distance.

Dans ce projet, nous exploitons les signaux DTMF comme méthode de commande pour allumer des LEDs. Le principe consiste à capter le signal audio à l'aide d'un microphone amplifié, à identifier la touche correspondante grâce à un traitement numérique, puis à activer la LED associée.

Ce projet permet de mettre en pratique des compétences en traitement du signal, en électronique et en programmation embarquée. Il constitue une excellente application pédagogique, combinant théorie des DSP (Digital Signal Processing) et mise en œuvre matérielle via un microcontrôleur.

Objectifs du projet

L'objectif principal est de concevoir un système capable de décoder les signaux DTMF pour commander l'allumage de LEDs correspondant aux touches pressées. Le signal audio est capté par un microphone amplifié (MAX4466), puis traité par un microcontrôleur qui effectue un décodage logiciel et allume la LED appropriée.

Ce projet illustre l'utilisation concrète des signaux DTMF dans des systèmes embarqués et permet de mettre en œuvre l'ensemble de la chaîne de traitement : **réception, décodage et commande**.

Utilisation concrète des signaux DTMF

- Permettent le contrôle d'appareils électroniques à distance via un téléphone.
- Utilisés dans les systèmes de menus vocaux interactifs (IVR).
- Employés en sécurité et domotique pour déclencher des actions à distance.
- Intégrés dans des systèmes embarqués pour l'automatisation de tâches simples.

Dans le cadre de ce projet

- Les signaux DTMF sont utilisés pour détecter les touches pressées.
- La touche reçue est indiquée par l'allumage d'une LED correspondante.
- Cela permet une vérification visuelle simple du bon fonctionnement du système.

Théorie

Qu'est-ce que le Le microphone MAX4466 ?

Le microphone MAX4466 est un microphone à condensateur (électret) équipé d'un amplificateur intégré. Il convertit les signaux acoustiques (les sons) en un signal électrique analogique amplifié, ce qui facilite leur traitement par un microcontrôleur.

Fonctionnement du MAX4466

- Le microphone capte les ondes sonores, qui font vibrer une membrane interne.
- Ces vibrations modifient la capacité électrique d'un condensateur interne, générant ainsi un petit signal électrique proportionnel à la pression acoustique.
- Ce signal est ensuite amplifié directement à l'intérieur du module MAX4466 par un amplificateur opérationnel intégré, ce qui améliore la qualité et la puissance du signal de sortie.
- Le niveau de gain de cet amplificateur est souvent réglable via un potentiomètre intégré sur la carte MAX4466, permettant d'adapter la sensibilité du microphone selon l'application.

Utilisation dans notre projet

Dans notre projet, le microphone MAX4466 sert à capturer les signaux audio DTMF provenant d'un téléphone ou d'un générateur de tonalités. Le signal amplifié est ensuite envoyé vers l'entrée analogique du microcontrôleur pour être traité et décodé en logiciel.

Théorie

Qu'est-ce que le DTMF ?

Le DTMF (Dual Tone Multi-Frequency), ou numérotation multiréquence, est une méthode de transmission de données utilisée notamment dans les systèmes téléphoniques. Chaque touche d'un clavier téléphonique envoie un signal composé de deux fréquences simultanées :

- Une fréquence provenant d'un groupe de fréquences basses,
- Et une autre d'un groupe de fréquences hautes.

Cette combinaison unique permet d'identifier chaque touche de manière fiable

Principe des doubles fréquences

Le clavier DTMF est organisé en une matrice 4x4 (dans les cas complets), avec chaque ligne et chaque colonne associées à une fréquence spécifique. Lorsque l'utilisateur appuie sur une touche, **le système génère une fréquence basse + une fréquence haute**.

Voici un tableau typique de correspondance :

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Méthodes de décodage DTMF

Pour interpréter les signaux DTMF, deux approches principales sont utilisées : **le décodage matériel et le décodage logiciel**.

1. Décodage matériel

Cette méthode repose sur l'utilisation d'un circuit intégré dédié, comme le MT8870, spécialement conçu pour le décodage des signaux DTMF.

- **Fonctionnement :**

- Le signal audio DTMF est transmis à l'entrée du MT8870.
- Le circuit effectue une détection automatique des deux fréquences présentes dans le signal.
- Il identifie la touche correspondante et fournit un code binaire à 4 bits en sortie, représentant cette touche.
- Ce code peut ensuite être lu par un microcontrôleur pour déclencher des actions (par exemple, allumer une LED).

- **Avantages :**

- Simplicité d'implémentation.
- Précision et fiabilité du décodage.
- Moins de charge pour le microcontrôleur.

- **Inconvénients :**

- Moins flexible (ne permet pas d'analyser ou manipuler les signaux).
- Dépendance au composant matériel (MT8870 ou équivalent).

2. Décodage logiciel

Le décodage logiciel repose sur l'analyse numérique du signal DTMF par un microcontrôleur. La méthode la plus courante est l'utilisation de la transformée de Fourier rapide (FFT).

- **Fonctionnement :**

- Le microcontrôleur lit le signal analogique à l'aide d'un convertisseur analogique/numérique (ADC).
- Il applique un algorithme FFT pour transformer le signal du domaine temporel au domaine fréquentiel.
- Les pics de fréquence détectés sont comparés aux fréquences DTMF standards pour identifier la touche appuyée.

- **Avantages :**

- Grande flexibilité.
- Pas besoin de circuit externe (tout est traité par le code).
- Permet une analyse fine du signal.

- **Inconvénients :**

- Plus complexe à implémenter.
- Nécessite plus de ressources de calcul (puissance de traitement et mémoire).
- Moins rapide que le décodage matériel si le microcontrôleur est limité.

Décodage logiciel dans notre projet : l'algorithme de Goertzel

Pour ce projet, nous avons choisi un décodage logiciel basé sur l'algorithme de Goertzel, qui est particulièrement adapté à la détection de quelques fréquences spécifiques, comme celles du DTMF.

Pourquoi l'algorithme de Goertzel ?

- Moins gourmand en ressources que la FFT.
- Idéal pour des microcontrôleurs avec des capacités limitées.
- Précis et efficace pour les huit fréquences du DTMF.

Étapes du traitement :

1. Le signal capté par le microphone est échantillonné via l'ADC du microcontrôleur.
2. L'algorithme de Goertzel calcule l'énergie de chaque fréquence DTMF.
3. Les deux fréquences dominantes sont identifiées.
4. La combinaison est associée à la touche correspondante.
5. Une LED est allumée selon la touche détectée.

Ce décodage logiciel nous a permis de simplifier le matériel nécessaire tout en assurant un traitement précis et rapide des signaux DTMF reçus.

Pourquoi avons-nous choisi le décodage Logiciel ?

Dans le cadre de notre projet, nous avons opté pour une solution de décodage logiciel des signaux DTMF, au lieu d'utiliser un décodage matériel via un circuit spécialisé tel que le MT8870. Ce choix s'appuie sur plusieurs considérations théoriques, pratiques et pédagogiques.

1. Maîtrise du traitement numérique du signal (DSP)

Le décodage logiciel permet de mettre en œuvre concrètement les principes du traitement numérique des signaux (Digital Signal Processing). En utilisant des algorithmes tels que Goertzel, nous avons pu :

- Travailler directement sur les échantillons du signal audio numérisé.
- Déetecter les composantes fréquentielles caractéristiques des signaux DTMF.
- Appliquer des techniques mathématiques sur le microcontrôleur pour interpréter les signaux reçus.

Cela nous a permis de renforcer notre compréhension des concepts comme la transformée de Fourier, la décomposition fréquentielle, la fenêtrage, ou encore la reconstruction d'un signal à partir de ses composantes.

2. Souplesse et adaptabilité de l'approche logicielle

Contrairement à une solution matérielle fixe, le décodage logiciel offre une grande flexibilité de paramétrage :

- Il est possible de modifier les seuils de détection des fréquences selon le bruit ambiant.
- L'algorithme peut être adapté pour ignorer certaines interférences ou fausses détections.
- De nouvelles fonctionnalités peuvent être intégrées facilement (affichage LCD, enregistrement, filtrage avancé, etc.).

Cela rend le système plus évolutif, ce qui est idéal dans un cadre d'expérimentation ou de prototypage.

3. Réduction des coûts et simplification matérielle

L'approche logicielle permet de se passer de composants dédiés comme le MT8870. Ainsi :

- Le circuit est simplifié, ce qui réduit les risques d'erreurs de câblage.
- Le coût global du projet est diminué.
- L'encombrement de la carte électronique est réduit, ce qui est utile pour des systèmes embarqués compacts.

Cela rend le projet plus accessible et facilite sa reproductibilité.

4. Optimisation pour systèmes embarqués modernes

Les microcontrôleurs récents (comme ceux de la famille ARM Cortex-M ou même les AVR plus puissants) disposent de suffisamment de puissance de calcul pour exécuter efficacement des algorithmes comme Goertzel, tout en continuant à gérer d'autres tâches (affichage, communication, commande de périphériques, etc.).

Le décodage logiciel devient donc une solution performante, viable et temps réel, même sur des plateformes embarquées à faibles ressources, tant que l'algorithme est bien optimisé.

5. Enrichissement pédagogique

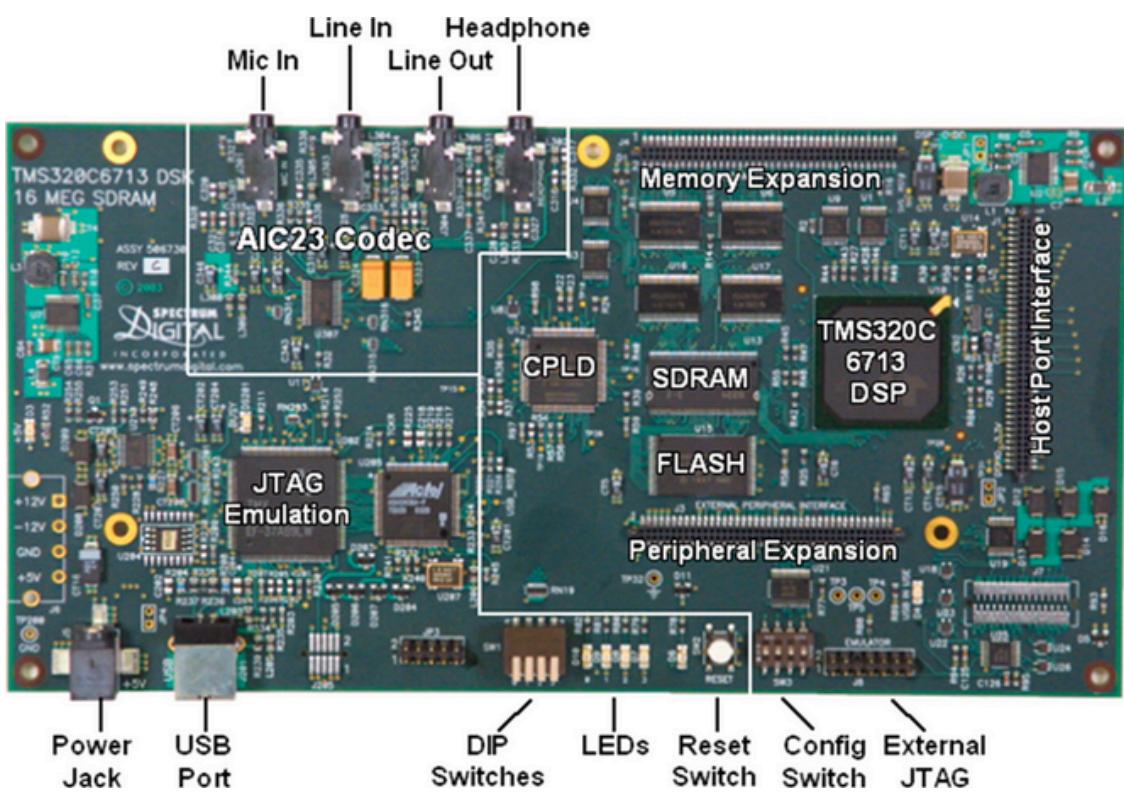
Ce choix s'inscrit aussi dans une logique d'apprentissage actif :

- Il oblige à manipuler des concepts avancés comme l'échantillonnage, la fréquence de Nyquist, les filtres numériques, ou encore le bruit.
- Il développe la capacité à résoudre des problèmes pratiques (détection dans un environnement bruité, ajustement des fenêtres d'analyse, etc.).

Matériel et Outils

Matériel Nécessaire

- Carte DSP DSK6713 de Texas Instruments.
- Codec audio intégré TLV320AIC23.
- Microphone ou générateur de tonalités DTMF.
- Câbles audio (jack 3.5 mm ou RCA).
- Logiciel Code ComposerStudio (CCS)



Implémentation

Code

```
#include <math.h> #include "dsk6713.h"
#include "dsk6713_aic23.h" #include "dsk6713_led.h"

// === Configuration de base du codec audio AIC23 ===
DSK6713_AIC23_CodecHandle hCodec;
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;

// === Déclarations === #define PI 3.141592653589793
#define SAMPLE_RATE 8000
#define N 205 // Nombre d'échantillons pour 25 ms à 8 kHz
short buffer[N]; short index = 0; int i=0;

float dtmf_freqs[] = {697, 770, 852, 941, 1209, 1336, 1477, 1633};
char dtmf_keys[4][4] = {
{'1','2', '3','A'},
{'4','5', '6','B'},
{'7','8', '9','C'},
{'*','0', '#','D'}
};

// === Prototypes ===
void set_gpio(int pin, int value); void display_dtmf_binary(char key);
float goertzel(short *data, int numSamples, float target_freq, int
'>     sample_rate);
short get_sample();

// === Fonctions de gestion des led===
void set_gpio(int pin, int value) { if (value)
    DSK6713_LED_on(pin); else
    DSK6713_LED_off(pin);
}
```

```
// === Activation des LED en fonction du signal détecté
void display_dtmf_binary(char key) { int value = -1;

    switch (key) {
        case '0': value = 0x0; break;
        case '1': value = 0x1; break;
        case '2': value = 0x2; break;
        case '3': value = 0x3; break;
        case '4': value = 0x4; break;
        case '5': value = 0x5; break;
        case '6': value = 0x6; break;
        case '7': value = 0x7; break;
        case '8': value = 0x8; break;
        case '9': value = 0x9; break;
        case 'A': value = 0xA; break;
        case 'B': value = 0xB; break;
        case 'C': value = 0xC; break;
        case 'D': value = 0xD; break;
        case '*': value = 0xF; break;
        case '#': value = -1; break;
        default:
            value = -1; break;
    }

    if (value >= 0) {
        set_gpio(0, value & 0x01); set_gpio(1, (value >> 1) & 0x01); set_gpio(2,
        (value >> 2) & 0x01); set_gpio(3, (value >> 3) & 0x01);
    } else {
        set_gpio(0, 0);
        set_gpio(1, 0);
        set_gpio(2, 0);
        set_gpio(3, 0);
    }
}
```

```

// --- Algorithme de Goertzel pour le traitement du signal ---
float goertzel(short *data, int numSamples, float target_freq, int
'> sample_rate) { int i, k;
float floatnumSamples = (float)numSamples;
float omega, sine, cosine, coeff, q0 =0,q1 =0,q2 =0; float real, imag,
result;

k= (int)(0.5 + ((floatnumSamples * target_freq) /sample_rate)); omega
= (2.0 * PI* k) / floatnumSamples;
sine =sin(omega); cosine = cos(omega); coeff =2.0*cosine;
for(i =0; i < numSamples; i++) { q0 =coeff * q1 - q2 + data[i]; q2 =q1;
q1=q0;
}
real =(q1 - q2 *cosine); imag =(q2 * sine);
result =sqrtf(real * real + imag *imag); return result;
}

// == Détection des fréquences ==
void detect_dtmf() { float magnitudes[8];
for (i =0; i<8;i++){
magnitudes[i] = goertzel(buffer, N, dtmf_freqs[i],
'> SAMPLE_RATE);
}
int row = -1, col =-1;
float max_low = 0,max_high =0;

for (i =0; i<4;i++){
if (magnitudes[i] > max_low) { max_low = magnitudes[i]; row = i;
}
}
for (i =4; i<8;i++){
if (magnitudes[i] > max_high) { max_high =magnitudes[i]; col = i - 4;
}
}
if (max_low > 1000 && max_high >1000) { charkey =dtmf_keys[row]
[col]; display_dtmf_binary(key);
}
}

```

```
// === Fonction pour lire audio par le codec AIC23
short get_sample() { short sample;
while (!MCBSP_rrdy(DSK6713_AIC23_DATAHANDLE)); // Wait for data
'→    ready
sample = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
// Read sample
'→    (16-bit)
return sample;
}

// === Fonction d'initialisation système ===
void init_system() {

// Initialisation de la carte DSK6713
DSK6713_init();

//Initialisation des GPIO
DSK6713_LED_init();

// Ouverture du codec avec la configuration par défaut
hCodec = DSK6713_AIC23_openCodec(0, &config);

// Réglage de la fréquence d'échantillonnage à 8 kHz
DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);
}

// === Main ===
void main() {
init_system();

while (1) {
for (index = 0; index < N; index++) { buffer[index] =get_sample();
}
detect_dtmf();
}
}
```

```
// === Fonction pour lire audio par le codec AIC23
short get_sample() { short sample;
while (!MCBSP_rrdy(DSK6713_AIC23_DATAHANDLE)); // Wait for data
'→    ready
sample = MCBSP_read(DSK6713_AIC23_DATAHANDLE);
// Read sample
'→    (16-bit)
return sample;
}

// === Fonction d'initialisation système ===
void init_system() {

// Initialisation de la carte DSK6713
DSK6713_init();

//Initialisation des GPIO
DSK6713_LED_init();

// Ouverture du codec avec la configuration par défaut
hCodec = DSK6713_AIC23_openCodec(0, &config);

// Réglage de la fréquence d'échantillonnage à 8 kHz
DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);
}

// === Main ===
void main() {
init_system();

while (1) {
for (index = 0; index < N; index++) { buffer[index] = get_sample();
}
detect_dtmf();
}
}
```

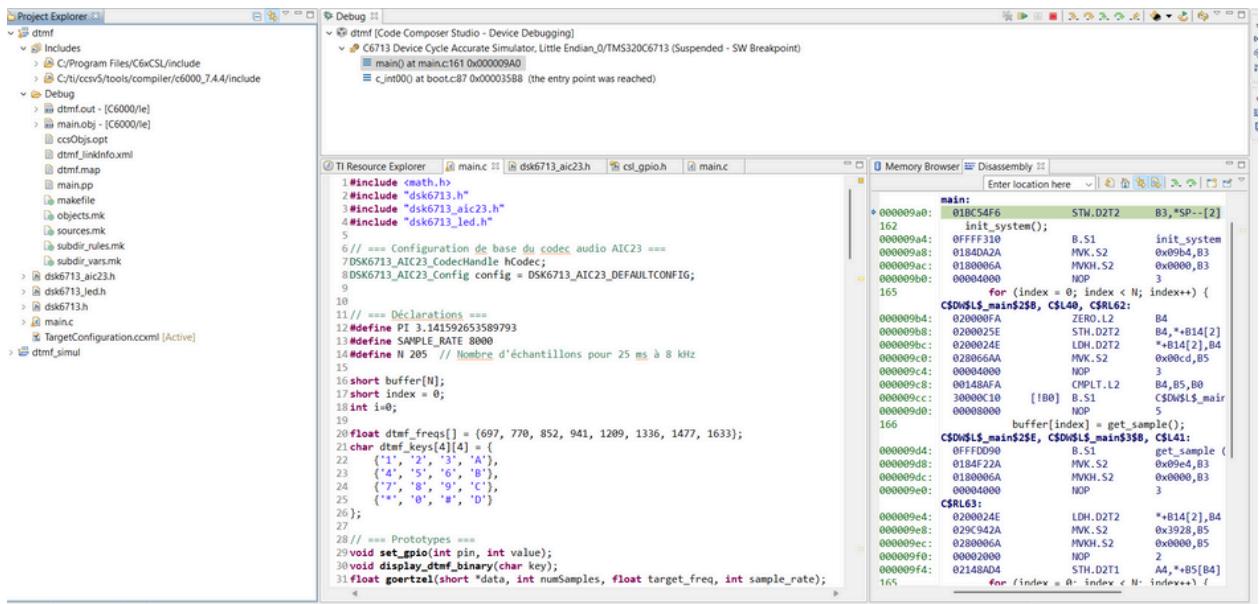


Figure Débogage et Build avec succès sur CCS

Explication du Code et Détails d'Implémentation

Initialisation

- DSK6713_init() initialise la carte DSP.
- DSK6713_LED_init() initialise les LED.
- DSK6713_AIC23_openCodec() ouvre la communication avec le codec audio.
- DSK6713_AIC23_setFreq() configure la fréquence d'échantillonnage à 8 kHz.

Lecture Audio

La fonction get_sample() utilise l'interface McBSP pour lire les données échantillonées par le codec :

```
short get_sample () {
    while (! MCBSP_rrdy ( DSK 6713 _AIC 23 _DATAHANDLE ));
    return MCBSP_read ( DSK 6713 _AIC 23 _DATAHANDLE );
}
```

Algorithme de Goertzel

Utilisé pour mesurer la puissance d'une fréquence cible dans un bloc d'échantillons :

```
float goertzel( short * data , int num Samples , float target_freq , int sample_rate ) {
    ...
    for (i = 0; i < num Samples; i++) { q0 = coeff * q1 -q2 + data [ i]; q2 = q1 ;
    q1 = q0 ;
    }
    ...
    result = sqrtf( real * real + imag * imag );
    return result;
}
```

Détection de Touche DTMF

detect_dtmf() applique Goertzel aux 8 fréquences DTMF. Elle identifie la fréquence basse et la fréquence haute avec l'amplitude maximale, puis détermine la touche :

```
void detect_dtmf () {
    ...
    if ( max_low > 1000 && max_high > 1000) { char key = dtmf_keys[ row ][ col]; display_dtmf_binary (key );
    }
    }
```

Le seuil (ici 1000) permet d'éviter les fausses détections en cas de bruit.

Affichage du Résultat via les LED

display_dtmf_binary() convertit la touche détectée en valeur binaire et l'affiche sur les LED (4 bits) :

```
void display_dtmf_binary ( char key ) {
    ...
    set_gpio (0 , value & 0 x01 ); set_gpio (1 , ( value >> 1) & 0 x01 );
    set_gpio (2 , ( value >> 2) & 0 x01 ); set_gpio (3 , ( value >> 3) & 0 x01 );
}
```

Boucle Principale

La fonction main() lit un bloc de 205 échantillons et appelle la détection DTMF :

```
void main () {  
    init_system (); while (1) {  
        for (index =0; index < N; index++) { buffer[ index ] =get_sample ();  
        }  
        detect_dtmf ();  
    }  
}
```

Conclusion

Ce programme fournit une base solide pour la détection de signaux DTMF sur la plateforme DSK6713. Grâce à l'algorithme de Goertzel, il est capable d'identifier efficacement les tonalités même avec des ressources limitées. Ne disposant pas de matériel pour la simulation réelle, nous travaillerons sur une carte Arduino nano.

Résultats

Simulation en environnement réel

Comme mentionné précédemment, par manque de matériel adapté, nous utiliserons Arduino Nano, un microphone MAX4466 et un module MAX7219 pour afficher la touche DTMF détectée via des fréquences DTMF. Le microphone capte les signaux audio, le processeur (Arduino Nano) applique l'algorithme Goertzel pour détecter les deux fréquences correspondantes, puis affiche la touche DTMF correspondante sur un écran à LEDs MAX7219.

Matériel utilisé

Carte de Développement Arduino Nano

L'Arduino Nano est une petite carte de développement équipée d'un microcontrôleur ATmega328P. Elle est conçue pour être utilisée dans des projets électroniques qui nécessitent une petite taille et une grande flexibilité.

1. Caractéristiques principales

- Microcontrôleur ATmega328P.
- 14 broches d'entrée/sortie numériques.
- 8 broches d'entrée analogique.
- Interface USB pour la programmation et l'alimentation.
- Compatible avec de nombreuses bibliothèques logicielles Arduino pour une utilisation facile.

2. Rôle dans ce projet

L'Arduino Nano est utilisé comme processeur central pour capter les signaux audio du microphone, exécuter l'algorithme Goertzel et contrôler l'affichage des résultats sur un module MAX7219. Il permet de traiter les données et de piloter les autres composants de manière coordonnée.

Microphone MAX4466

Le MAX4466 est un microphone à condensateur amplifié conçu pour capter des signaux audio analogiques. Il est équipé d'un amplificateur intégré à faible bruit, ce qui le rend particulièrement sensible aux sons faibles. Il est utilisé dans de nombreux systèmes audio embarqués pour sa capacité à fournir des signaux clairs et précis.

1.Caractéristiques principales

- Microphone à condensateur avec amplification interne.
- Sortie analogique proportionnelle à l'intensité du son capté.
- Faible bruit de fond pour des applications précises.
- Alimentation via une tension de 3V à 12V.
- Sortie analogique permettant de le connecter facilement à des microcontrôleurs comme l'Arduino.

2.Rôle dans ce projet

Le microphone MAX4466 capte les tonalités DTMF environ-nantes (généralement émises par des téléphones ou d'autres appareils), et les convertit en signaux analogiques. Ces signaux sont ensuite envoyés à l'Arduino Nano pour être traités.

Module MAX7219 (Affichage 8x8 LED)

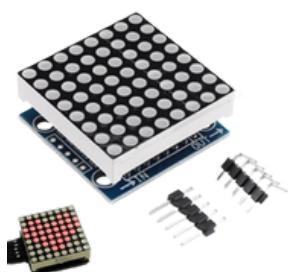
Le MAX7219 est un contrôleur de matrice de LEDs qui permet de piloter des affichages LED de manière simple et efficace. Il est couramment utilisé dans les projets Arduino pour afficher des caractères, chiffres ou graphiques sur des matrices d'affichage à LEDs.

1.Caractéristiques principales

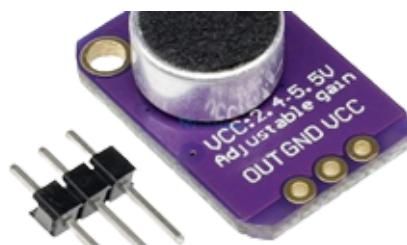
- Contrôle jusqu'à 8x8 LEDs (64 LEDs au total) avec un minimum de broches.
- Communication via une interface série (SPI).
- Facile à connecter à des microcontrôleurs comme l'Arduino.
- Compatible avec des matrices de LEDs ou des chiffres à LED.

2.Rôle dans ce projet

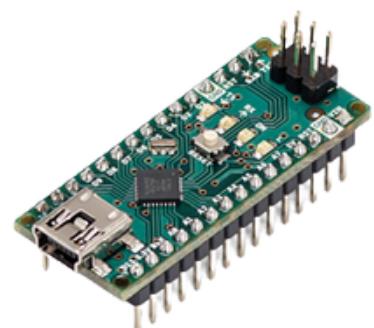
Le module MAX7219 est utilisé pour afficher visuellement les résultats de la détection DTMF. Une fois que l'Arduino identifie la touche DTMF correspondante, le module MAX7219 allume les LEDs de la matrice pour afficher la touche détectée, comme un chiffre ou un symbole.



Afficheur MAX7219

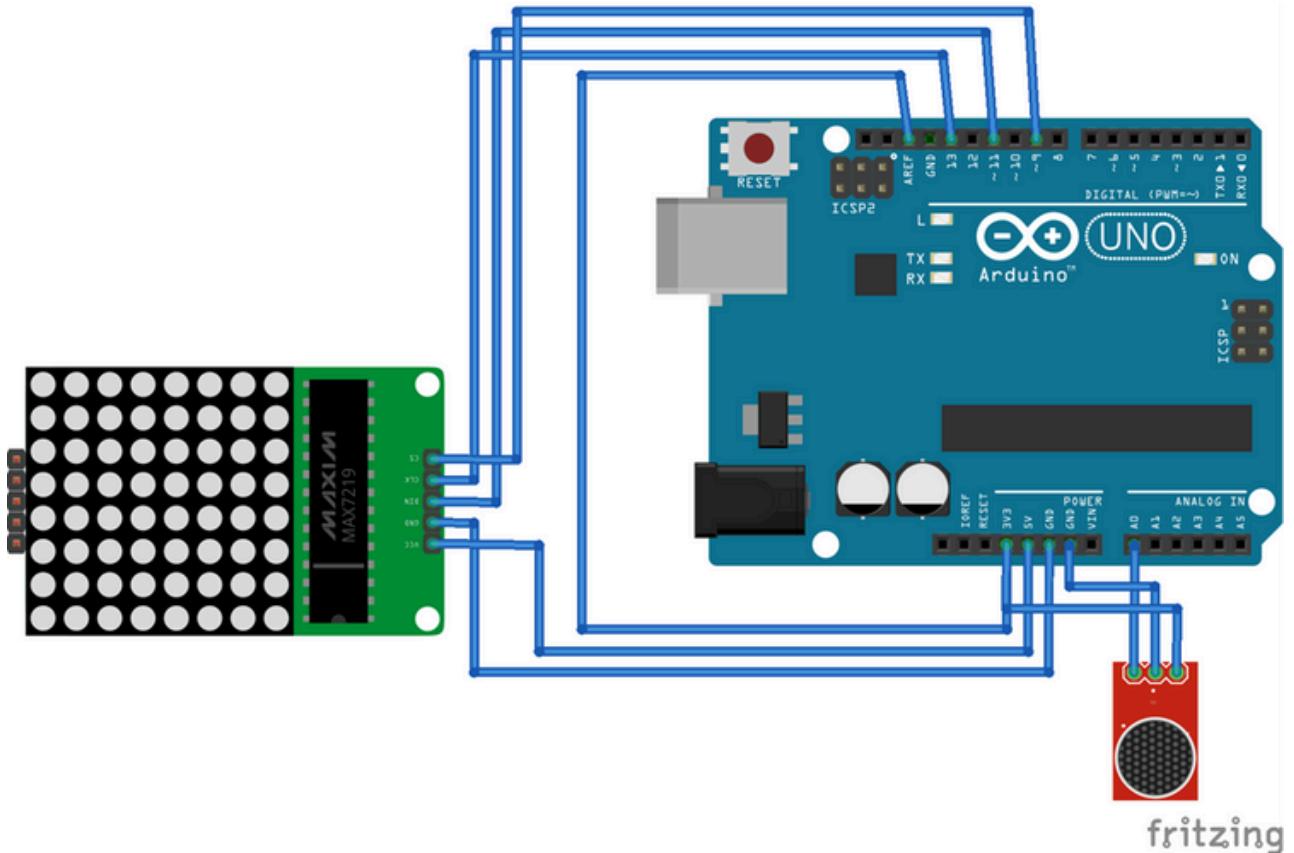


Microphone MAX4466



Arduino nano

Schéma du câblage



Code Arduino

```
#include <binary.h>
#include <avr/pgmspace.h>

#include <LEDMatrixDriver.hpp>

#define CS_PIN    9

#define N      256
#define IX_LEN  8
#define THRESHOLD 20

LEDMatrixDriver lmd(1, CS_PIN);
```

```
uint8_t samples[N];
volatile uint16_t samplePos = 0;

float spectrum[IX_LEN];

// Frequencies [697.0, 770.0, 852.0, 941.0, 1209.0, 1336.0, 1477.0,
1633.0]
// Calculated for 9615Hz 256 samples
const float cos_t[IX_LEN] PROGMEM = {
    0.8932243011955153, 0.8700869911087115, 0.8448535652497071,
    0.8032075314806449,
    0.6895405447370669, 0.6343932841636456,
    0.5555702330196023, 0.4713967368259978
};

const float sin_t[IX_LEN] PROGMEM = {
    0.44961132965460654, 0.49289819222978404,
    0.5349976198870972, 0.5956993044924334,
    0.7242470829514669, 0.7730104533627369, 0.8314696123025451,
    0.8819212643483549
};

typedef struct {
    char digit;
    uint8_t index;
} digit_t;

digit_t detected_digit;

const char table[4][4] PROGMEM = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

const uint8_t char_indexes[4][4] PROGMEM = {
    {1, 2, 3, 10},
    {4, 5, 6, 11},
    {7, 8, 9, 12},
    {15, 0, 14, 13}
};
```

```

byte font[16][8] = {
    {0x00,0x38,0x44,0x4c,0x54,0x64,0x44,0x38}, // 0
    {0x04,0x0c,0x14,0x24,0x04,0x04,0x04,0x04}, // 1
    {0x00,0x30,0x48,0x04,0x04,0x38,0x40,0x7c}, // 2
    {0x00,0x38,0x04,0x04,0x18,0x04,0x44,0x38}, // 3
    {0x00,0x04,0x0c,0x14,0x24,0x7e,0x04,0x04}, // 4
    {0x00,0x7c,0x40,0x40,0x78,0x04,0x04,0x38}, // 5
    {0x00,0x38,0x40,0x40,0x78,0x44,0x44,0x38}, // 6
    {0x00,0x7c,0x04,0x04,0x08,0x08,0x10,0x10}, // 7
    {0x00,0x3c,0x44,0x44,0x38,0x44,0x44,0x78}, // 8
    {0x00,0x38,0x44,0x44,0x3c,0x04,0x04,0x78}, // 9
    {0x00,0x1c,0x22,0x42,0x42,0x7e,0x42,0x42}, // A
    {0x00,0x78,0x44,0x44,0x78,0x44,0x44,0x7c}, // B
    {0x00,0x3c,0x44,0x40,0x40,0x40,0x44,0x7c}, // C
    {0x00,0x7c,0x42,0x42,0x42,0x42,0x44,0x78}, // D
    {0x00,0x0a,0x7f,0x14,0x28,0xfe,0x50,0x00}, // #
    {0x00,0x10,0x54,0x38,0x10,0x38,0x54,0x10} /* */
};

void initADC() {
    // Init ADC; f = ( 16MHz/prescaler ) / 13 cycles/conversion
    ADMUX = 0; // Channel sel, right-adj, use AREF pin
    ADCSRA = _BV(ADEN) | // ADC enable
        _BV(ADSC) | // ADC start
        _BV(ADATE) | // Auto trigger
        _BV(ADIE) | // Interrupt enable
        _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0); // 128:1 / 13 = 9615 Hz
    ADCSRB = 0; // Free-run mode
    DIDR0 = _BV(0); // Turn off digital input for ADC pin
    TIMSK0 = 0; // Timer0 off
}

```

```
void goertzel(uint8_t *samples, float *spectrum) {
    float v_0, v_1, v_2;
    float re, im, amp;

    for (uint8_t k = 0; k < IX_LEN; k++) {
        float c = pgm_read_float(&(cos_t[k]));
        float s = pgm_read_float(&(sin_t[k]));

        float a = 2. * c;
        v_0 = v_1 = v_2 = 0;
        for (uint16_t i = 0; i < N; i++) {
            v_0 = v_1;
            v_1 = v_2;
            v_2 = (float)(samples[i]) + a * v_1 - v_0;
        }
        re = c * v_2 - v_1;
        im = s * v_2;
        amp = sqrt(re * re + im * im);
        spectrum[k] = amp;
    }
}

float avg(float *a, uint16_t len) {
    float result = .0;
    for (uint16_t i = 0; i < len; i++) {
        result += a[i];
    }
    return result / len;
}

int8_t get_single_index_above_threshold(float *a, uint16_t len, float threshold) {

    if (threshold < THRESHOLD) {
        return -1;
    }
    int8_t ix = -1;
    for (uint16_t i = 0; i < len; i++) {
        if (a[i] > threshold) {
            if (ix == -1) {
                ix = i;
            } else {
                return -1;
            }
        }
    }
    return ix;
}
```

```
void detect_digit(float *spectrum) {
    float avg_row = avg(spectrum, 4);
    float avg_col = avg(&spectrum[4], 4);
    int8_t row = get_single_index_above_threshold(spectrum, 4,
    avg_row);
    int8_t col = get_single_index_above_threshold(&spectrum[4], 4,
    avg_col);

    if (row != -1 && col != -1 && avg_col > 200) {
        detected_digit.digit = pgm_read_byte(&(table[row][col]));
        detected_digit.index = pgm_read_byte(&(char_indexes[row]
        [col]));
    } else {
        detected_digit.digit = 0;
    }
}

void drawSprite(byte* sprite) {
    byte mask = B10000000;

    for(int iy = 0; iy < 8; iy++) {
        for(int ix = 0; ix < 8; ix++) {
            lmd.setPixel(7 - iy, ix, (bool)(sprite[iy] & mask));
            mask = mask >> 1;
        }
        mask = B10000000;
    }
}

void setup() {
    cli();
    initADC();
    sei();

    Serial.begin(115200);

    lmd.setEnabled(true);
    lmd.setIntensity(2);
    lmd.clear();
    lmd.display();

    detected_digit.digit = 0;
}

unsigned long z = 0;
```

```
void loop() {
    while(ADCSRA & _BV(ADIE)); // Wait for audio sampling to finish
    goertzel(samples, spectrum);
    detect_digit(spectrum);

    if (detected_digit.digit != 0) {
        drawSprite(font[detected_digit.index]);
        lmd.display();
    }

    if (z % 5 == 0) {
        for (int i = 0; i < IX_LEN; i++) {
            Serial.print(spectrum[i]);
            Serial.print("\t");
        }
        Serial.println();
        Serial.println((int)detected_digit.digit);
    }
    z++;

    samplePos = 0;

    ADCSRA |= _BV(ADIE); // Resume sampling interrupt
}

ISR(ADC_vect) {
    uint16_t sample = ADC;

    samples[samplePos++] = sample - 400;

    if(samplePos >= N) {
        ADCSRA &= ~_BV(ADIE); // Buffer full, interrupt off
    }
}
```

Explication du code

- Module MAX4466 (Microphone) : Le microphone MAX4466 est utilisé pour capter des échantillons audio à partir de l'environnement. Les données sont lues à partir de la pin analogique A0 sur l'Arduino.
- Algorithme Goertzel : L'algorithme Goertzel est utilisé pour détecter la présence des 8 fréquences DTMF.
- Affichage sur MAX7219 : Le module MAX7219 est utilisé pour afficher le caractère correspondant à la touche DTMF détectée.
- Détection et Affichage : Si les magnitudes des fréquences détectées sont suffisamment fortes (seuil de 10,000), la touche correspondante est affichée sur le module MAX7219. Sinon, il n'y a pas de détection valide.

Applications pratiques

Les tonalités DTMF (Dual-Tone Multi-Frequency) sont largement utilisées dans diverses applications pour transmettre des informations via des signaux audio dont les deux suivantes :

1.Systèmes de télécommande utilisant des tonalités DTMF

Les tonalités DTMF sont couramment utilisées dans des systèmes de télécommande à distance, notamment pour contrôler des appareils ou des systèmes via un réseau téléphonique. Le principe repose sur l'utilisation des tonalités comme signal de commande, permettant d'interagir avec des systèmes à distance sans nécessiter une connexion de données complexe.

Applications

- Contrôle de la sécurité domestique : Un utilisateur peut désactiver une alarme ou ouvrir une porte verrouillée en composant un numéro DTMF depuis un téléphone.
- Télécommande d'appareils électroniques : Les tonalités DTMF servent à envoyer des commandes précises pour allumer, éteindre ou ajuster des paramètres d'appareils comme des climatiseurs, des dispositifs de chauffage ou des machines industrielles.

Avantages

- Simplicité et faible coût : Le système DTMF est simple à mettre en place et utilise les infrastructures téléphoniques existantes.
- Accessibilité : N'importe quel téléphone, fixe ou mobile, permet d'interagir avec le système.

2.Systèmes de saisie par clavier téléphonique

Les tonalités DTMF sont également utilisées dans les systèmes de saisie par clavier téléphonique, une application courante dans les services automatisés, les systèmes bancaires, et d'autres services nécessitant une interaction avec un utilisateur.

Applications

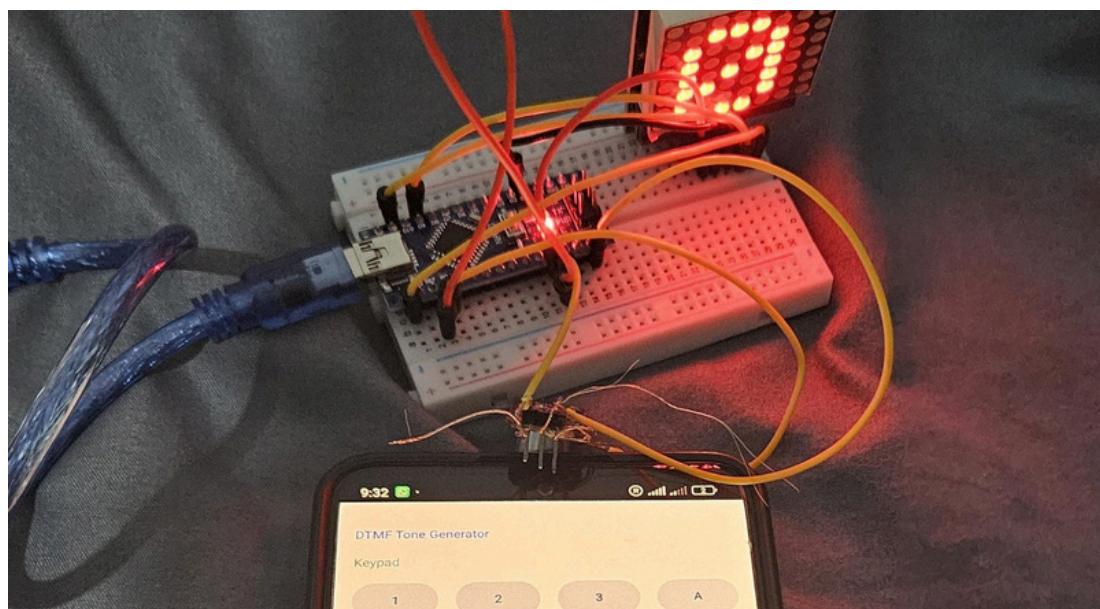
- Banque à distance : Les clients peuvent consulter leur solde, effectuer des virements ou accéder à l'historique des transactions.
- Systèmes d'annuaires téléphoniques automatisés : L'utilisateur accède à des informations (ex : restaurants, services) via des menus DTMF.
- Systèmes de messagerie vocale: Les utilisateurs naviguent dans les menus, écoutent ou suppriment des messages vocaux.

Avantages

- Autonomie : Les utilisateurs accèdent à des services sans intervention humaine, ce qui améliore l'efficacité.
- Multiplateforme : Compatible avec tout type de téléphone.
- Rapidité et simplicité : Une simple pression sur les touches suffit pour obtenir un résultat immédiat.

Conclusion

Une vidéo présentant le résultat de la simulation est jointe dans le dossier du projet.



Conclusion

Dans le cadre de notre projet intitulé « Implémentation d'un système pour décoder les signaux DTMF et allumer des LED correspondantes », nous avons développé une solution efficace permettant d'identifier les tonalités DTMF à l'aide d'un décodage logiciel, basé sur l'algorithme de Goertzel. Ce choix nous a permis d'éviter l'utilisation de composants matériels spécialisés comme le MT8870, tout en assurant une reconnaissance fiable des fréquences duales associées à chaque touche du clavier.

Le décodage logiciel présente plusieurs avantages clés : il offre une grande flexibilité d'adaptation, réduit les coûts en supprimant les composants dédiés, et nous a permis de mieux comprendre le fonctionnement du traitement numérique du signal dans un contexte embarqué. L'utilisation du microphone MAX4466, couplé au convertisseur analogique-numérique du microcontrôleur, nous a permis de capter et d'analyser efficacement les signaux audio en temps réel.

Cette approche nous a permis d'allumer des LEDs correspondant aux touches détectées, traduisant de manière visuelle et immédiate l'action de l'utilisateur.

Parmi les améliorations possibles à l'avenir :

- Ajouter un écran LCD pour afficher la touche reconnue.
- Implémenter une communication sans fil (Wi-Fi, Bluetooth).
- Développer une interface mobile pour enrichir les fonctionnalités de commande à distance.

En résumé, ce projet nous a permis de consolider nos compétences en électronique, programmation embarquée, et traitement du signal numérique, tout en atteignant l'objectif principal : concevoir un système interactif, réactif et adaptable basé sur la détection des tonalités DTMF.

