Université de Cergy-Pontoise

Rapport

Pour le projet UE Introduction aux microcontrôleurs

Licence d'Informatique troisième année

Sur le sujet

Implémentation d'un système de communication sans fil multimodalités entre microcontrôleurs STM32F4

Rédigé Par :

WATIER Antonin, BOUKILI Inas, HIMRANE Nehla,

JAHOUH Said, OFFIONG Dara, MEZIANE Fella, ABDELBASSIR Imane

Groupe C3

17 mars 2025

Introduction

1. Contexte et objectifs du projet

Ce projet vise à implémenter un système de communication sans fil entre deux microcontrôleurs **STM32F4**, utilisant deux modalités :

- **Communication sonore** (via un buzzer et un microphone).
- **Communication infrarouge** (via une LED IR et un récepteur).

L'objectif est de transmettre des messages entre deux ordinateurs en convertissant le texte en Morse, puis en l'envoyant sous forme de signaux sonores ou lumineux.

Description du projet

2. Principe de fonctionnement

Le système repose sur une chaîne de traitement en 4 étapes :

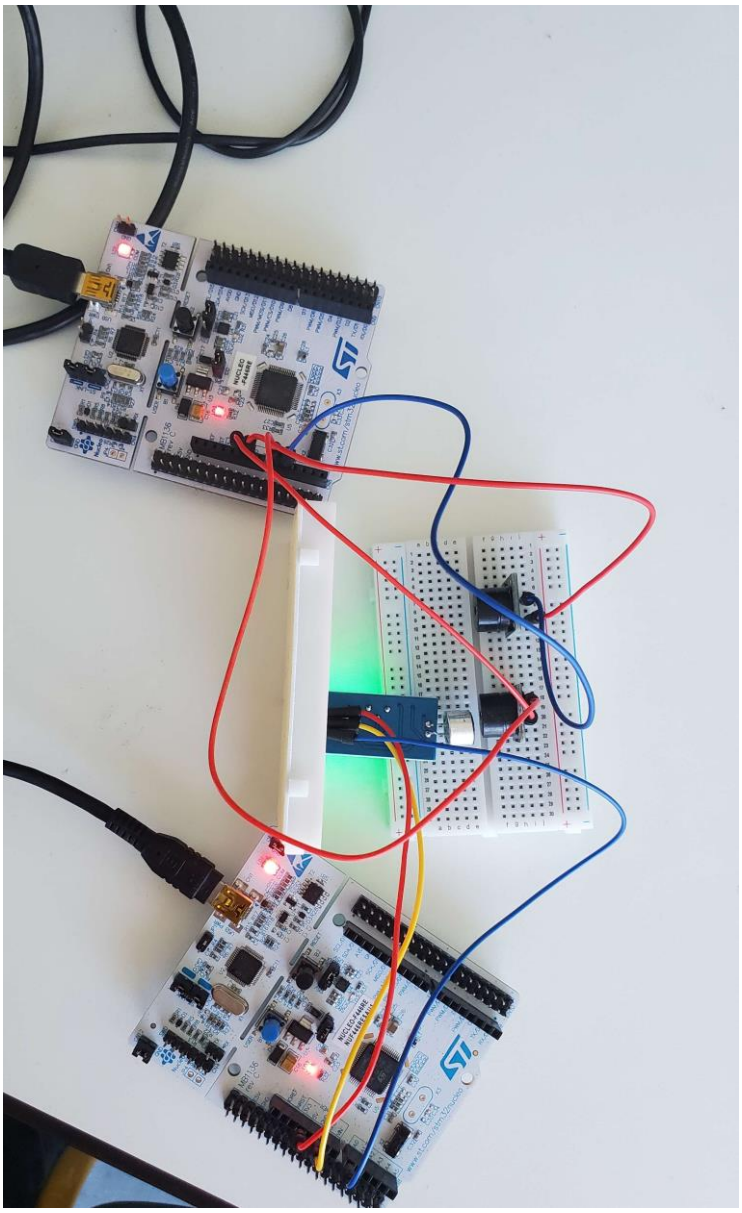
- **Saisie du message** : Un PC envoie un texte via UART à l'émetteur (STM32).
- **Encodage Morse** : Le microcontrôleur convertit le texte en séquences de points (courts) et tirets (longs).
- **Transmission sans fil** :
  a. **Modalité sonore** : Le buzzer émet des signaux PWM modulés en durée.
  b. *(Optionnel : Modalité infrarouge si implémentée)*.
- **Réception et décodage** :
  c. Le microphone capte le signal, l'ADC le numérise.
  d. Le récepteur mesure les durées pour reconstituer le Morse, puis le convertit en texte.

3. Schéma général du système

La figure ci-dessous illustre l'architecture générale du système de communication sans fil entre les deux microcontrôleurs STM32F4. Ce système repose sur deux modes de transmission :

- **Communication sonore** : Un buzzer génère des signaux en code Morse qui sont captés par un microphone, permettant ainsi l'échange d'informations sous forme de signaux acoustiques.
- **Connexion et traitement** : Chaque microcontrôleur est programmé pour encoder et décoder les signaux afin d'assurer la transmission et la réception des messages.

L'image suivante présente le câblage et l'interconnexion des différents composants :



3. Composants matériels et logiciels utilisés

3.1. Matériel

Le tableau ci-dessous résume les principaux composants matériels et logiciels utilisés dans notre système ainsi que leur rôle dans le projet.

Catégorie	Composant	Rôle
Microcontrôleurs	STM32F4 (x2)	Assurent le traitement des signaux et la communication entre les modules
Capteur audio	Microphone	Capte le signal sonore transmis en code Morse
Émission sonore	Buzzer	Génère les signaux sonores correspondant au code Morse

Carte de prototypage	Breadboard	Permet les connexions entre les différents composants
Câblage	Fils de connexion	Assurent l'interconnexion des modules
Alimentation	USB (via PC)	Alimente les microcontrôleurs
Logiciels	STM32CubeIDE	Programmation et développement du firmware
Librarie	HAL (Hardware Abstraction Layer)	Facilite la gestion des périphériques du microcontrôleur

3.2. Logiciels

- **STM32CubeIDE** : Développement du firmware (C).
- **STM32CubeMX** : Configuration des périphériques (GPIO, UART, PWM, ADC).
- **Terminal série** (TeraTerm) : Envoi/réception des messages.

Implémentation du système

4. Définition des schémas de connexion

3.1 Émetteur

Composant	Broche STM32	Configuration	Remarques
Buzzer1	PA0 (TIM2_CH1)	PWM (2 kHz) Prescaler : 84 –1 ARR : 500-1 Pulse: 250	
Buzzer2	PA1 (TIM2_CH2)	PWM (2 kHz) Prescaler : 84 –1 ARR : 500-1 Pulse: 250	
UART	USART2 (PA2-TX, PA3-RX)	Baudrate 9600, 8 bit de longueur, 1 bit de stop	Câble USB-TTL (FT232RL)
Alimentation	3.3V / GND	Alim carte Nucleo	-

3.2 Récepteur

Composant	Broche STM32	Configuration	Remarques
Microphone	PA0 (ADC1_IN0)	ADC 12 bits, Continuous conversion mode : Enable Sampling time : 15 cycles	Préampli LM358 si nécessaire
UART	USART2 (PA2-TX, PA3-RX)	Baudrate 115200, Prescaler : 84 –1 ARR : 500-1 Pulse: 250	Câble USB-TTL (FT232RL)
Bouton	PC13 (GPIO_EXTI13)	Rising/Falling edge Mode Pull-up (0 -> 3.3v)	
Alimentation	3.3V / GND	Alim carte Nucleo	-

1. **Buzzer_1** :
    a. Utilise **TIM2_CH1 (PA0)** en PWM (cf. `MX_TIM2_Init()` dans l'émetteur).
    b. Fréquence PWM réglée à 2 kHz
2. **Buzzer** 2:
    a. Utilise **TIM2_CH2 (PA1)** en PWM (cf. `MX_TIM2_Init()` dans l'émetteur).
    b. Fréquence PWM réglée à 2 kHz

3. **Microphone** :
    a. Connecté à **PA0 (ADC1_IN0)** (cf. `MX_ADC1_Init()` dans le récepteur).
    b. Seuil de détection réglable via `#define THRESHOLD 2000`.
4. **UART** :
    a. Les broches **PA2 (TX)** et **PA3 (RX)** sont utilisées pour les deux cartes (émetteur et récepteur).

3.3 Configuration de l'environnement de développement

Emetteur :

Configuration TIM2:

Counter Settings	
Prescaler (PSC - 16 bits value)	83
Counter Mode	Up
Counter Period (AutoReload Register - 32 bits value)	499
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (32 bits value)	250
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 2	
Mode	PWM mode 1
Pulse (32 bits value)	250
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

Configuration UART :

Basic Parameters	
Baud Rate	9600 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples

Récepteur:

Configuration ADC1 :

ADC_Settings	
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion
ADC_Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 0
Sampling Time	15 Cycles
ADC Injected ConversionMode	

Configuration TIM3 :

Counter Settings	
Prescaler (PSC - 16 bits value)	84 - 1
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	200 - 1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

Configuration UART :

Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples

5. Implémentation Logicielle

3.1. Émetteur (Encodage Morse)

Fonctions Clés

a. `HAL_UART_RxCpltCallback`

Rôle : Gère la réception asynchrone du message via UART.
Des qdu'une touche est entrée dans le terminal série, cette fonction est appelée

- Stocke les caractères dans `received_message si \n (fin du message)`.
- Sinon passe a l'index suivant de la chaine de caractère
- Ecrit dans reived_message le caractère.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
  if (huart->Instance == USART2) {
      if (received_message[message_index] == '\n' || message_index >=
MAX_MESSAGE_LENGTH - 1) {
          received_message[message_index] = '\0'; // Fin du message
          message_index = 0;
          message_received = true; // Prêt pour le traitement
      } else {
          message_index++;
      }
      HAL_UART_Receive_IT(&huart2, (uint8_t*)received_message + message_index,
1); // Réarmement
  }
}
```

b. `send_morse_message`

Rôle : Parcourt le message et appelle `send_morse_signal` pour chaque caractère.

- Gère les espaces (`WORD_GAP = 1000 ms`).
- Convertit les caractères en symboles Morse via les tables `morse_letter` et `morse_digit`.

```
void send_morse_message(const char* message) {
  for (int i = 0; message[i] != '\0'; i++) {
      char c = message[i];
      if (c == ' ') {
          wait_ms(INTER_WORD_GAP - INTER_CHARACTER_GAP); // Pause entre mots
      } else {
          const char* code = NULL;
          if (c >= 'A' && c <= 'Z') code = morse_letter[c - 'A'];
          else if (c >= '0' && c <= '9') code = morse_digit[c - '0'];

          if (code != NULL) {
              for (int j = 0; code[j] != '\0'; j++) {
```

```
                send_morse_signal(code[j]); // Émission du point/trait
                if (code[j+1] != '\0') wait_ms(INTER_ELEMENT_GAP); // Pause
entre éléments
            }
            wait_ms(INTER_CHARACTER_GAP); // Pause entre caractères
        }
    }
  }
  printf("Fin du message\r\n");
}
```

c. send_morse_signal

Rôle : Émet un son via le buzzer (PWM) pour un symbole Morse (`.` ou `-`).

- **Point (.)** : Durée courte (`DOT_DURATION = 150 ms`).
- **Trait (-)** : Durée longue (`DASH_DURATION = 400 ms`).

```
void send_morse_signal(char symbol) {
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // Activation PWM (buzzer)
  if (symbol == '.') wait_ms(DOT_DURATION);
  else if (symbol == '-') wait_ms(DASH_DURATION);
  HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1); // Désactivation PWM
}
```

d. wait_ms (Non-bloquant)

Rôle : Attend un délai en millisecondes sans bloquer le CPU. (Systick : timer intégré au microcontroleur).

- Utilise `SysTick_Handler` pour décrémenter `delay_time`.
- Permet de gérer le temps pour la durée d'un point, d'un tirer, d'un espace entre 2 caractère morses, d'un espace entre 2 caractères du message et d'un espace entre 2 mots du message.

```
volatile uint32_t delay_time = 0;

void SysTick_Handler(void) {
    HAL_IncTick();
    if (delay_time > 0) delay_time--; // Décrémentation du délai
}

void wait_ms(uint32_t ms) {
    delay_time = ms;
    while (delay_time > 0); // Attente active
}
```

3.2. Récepteur (Décodage Morse)

Fonctions Clés

a. HAL_TIM_PeriodElapsedCallback

Rôle : Détecte les signaux sonores toutes les 1 ms via l'ADC.
Logique :

1. Acquisition de la valeur ADC (seuil réglable via ADC_THRESHOLD).
2. Détection des transitions **son → silence** et inversement.
3. Si !is_singal : Mesure des durées pour distinguer **points (.)** et **traits (-)**.
4. Si is_signal : Mesure des durées pour distinguer les différents type d'espace (silence).

```
if (adc_value > ADC_THRESHOLD) {
  if (!ra.is_signal) {
      uint32_t silence_duration = now - last_time;
      process_silence_duration(silence_duration); // Traitement du silence
      ra.is_signal = true;
  }
} else {
  if (ra.is_signal) {
      uint32_t signal_duration = now - last_time;
      process_signal_duration(signal_duration); // Traitement du signal
      ra.is_signal = false;
  }
}
```

b. process_signal_duration

Rôle : Interprète la durée d'un signal pour déterminer s'il s'agit d'un **point (150 ms)** ou d'un **tiret (400 ms)**.

```
void process_signal_duration(uint32_t duration) {
  if (duration >= DOT_DURATION - 50 && duration < DOT_DURATION + 50) {
      ra.morse_symbol[ra.morse_index++] = '.'; // Point
  } else if (duration >= DASH_DURATION - 50 && duration < DASH_DURATION + 50)
{
      ra.morse_symbol[ra.morse_index++] = '-'; // Trait
  }
  printf("Symbole Morse: %s\r\n", ra.morse_symbol); // Debug
}
```

c. process_silence_duration

Rôle : Détecte les espaces entre caractères (`CHAR_GAP = 400 ms`) ou mots (`WORD_GAP = 1000 ms`), et déclenche le décodage.
Si il s'agit d'un espace entre 2 caractères morse, on ne fait rien (200 ms).

```
void process_silence_duration(uint32_t duration) {
  if (duration >= WORD_GAP - 100) {
      // Espace entre mots → Ajout d'un espace au message
      add_to_message(' ');
  } else if (duration >= CHAR_GAP - 50) {
      // Espace entre caractères → Décodage du symbole
      char c = decode_morse(ra.morse_symbol);
      add_to_message(c);
      ra.morse_index = 0; // Réinitialisation pour le prochain symbole
  }
}
```

d. decode_morse

Rôle : Convertit un symbole Morse (ex: `".-"`) en caractère alphanumérique (ex: `'A'`).
Méthode :
Comparaison avec les tables prédéfinies `morse_letter[]` et `morse_digit[]`.

```
char decode_morse(const char* symbol) {
  for (int i = 0; i < 26; i++) {
      if (strcmp(symbol, morse_letter[i]) == 0) return 'A' + i;
  }
  for (int i = 0; i < 10; i++) {
      if (strcmp(symbol, morse_digit[i]) == 0) return '0' + i;
  }
  return '?'; // Caractère inconnu
}
```

e. Gestion du Bouton (Interruption)

Rôle : Active/désactive l'enregistrement via le bouton **PC13**.

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if (GPIO_Pin == GPIO_PIN_13) {
      if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET) {
          init_audio(); // Démarre la capture
      } else {
          end_audio();  // Arrête et affiche le message final
          printf("Message reçu: %s\r\n", message);
      }
  }
}
```

Étude fonctionnelle et limitations

1. Distance de fonctionnement

Lors des tests réalisés, nous avons observé que le système fonctionne de manière optimale à une distance **modérée** entre l'émetteur (buzzer) et le récepteur (microphone).

- **Trop proche** : Aucun problème de transmission, mais risque de saturation du signal.
- **Trop loin** : La transmission devient moins fiable et le microphone peine à capter correctement le signal.
- **Distance idéale** : Une distance moyenne (max 10cm) assure un bon équilibre entre clarté et réception.

2. Sensibilité aux interférences et obstacles

Le système est **très sensible aux interférences**, ce qui peut impacter la qualité de la transmission :

- **Sources de bruit ambiant** : Les bruits environnants (parole, musique, autres signaux sonores) peuvent perturber la réception.
- **Réflexions sonores** : Dans un environnement fermé, les réflexions du son peuvent altérer la clarté du signal.

3. Performance du système (temps de transmission, fiabilité)

Les tests ont démontré que le système est **fonctionnel** et assure une transmission correcte des messages en Morse.

- **Temps de transmission** : Le temps de transmission dépend de la longueur du message, mais reste globalement rapide.

- **Fiabilité** : Si les conditions sont optimales (bonne distance, faible bruit ambiant), le système décode correctement les signaux et assure une transmission efficace. Cependant, la présence d'interférences peut introduire des erreurs.

Améliorations possibles

1. **Protocole Bidirectionnel** : Ajouter un **accusé de réception** via UART entre les deux STM32F4. Cela permet de confirmer la réception correcte des messages et d'éviter les erreurs.
2. **Utilisation d'un signal infrarouge** : Remplacer ou compléter la transmission sonore avec un **émetteur/récepteur infrarouge** (comme un module IR) pour éviter les interférences dues aux bruits environnants.
3. **Filtrage du signal audio** : Ajouter un **filtre passe-bande** sur le microphone pour isoler la fréquence du buzzer et réduire les interférences sonores.
4. **Encodage avancé des signaux** : Ajouter une **redondance ou correction d'erreur** (ex: code Hamming) pour rendre la transmission plus robuste face aux perturbations.

Conclusion

En conclusion, ce projet a permis d'explorer les principes de transmission et de réception de signaux acoustiques avec des **STM32F4**, tout en mettant en pratique des concepts de traitement du signal et de communication sans fil. Ces travaux ouvrent la voie à de futures optimisations et applications dans des domaines variés comme la transmission de données en environnements contraints ou la communication alternative pour des systèmes embarqués.

- ❖ Ce projet a permis de concevoir et d'implémenter un système de communication sans fil entre deux microcontrôleurs **STM32F4**, utilisant la transmission de signaux sonores en **code Morse**. À travers la conversion du texte en signaux sonores et leur réception via un microphone, nous avons pu établir un canal de communication fonctionnel entre deux ordinateurs.

- ❖ Les tests réalisés ont confirmé la faisabilité du système, avec une transmission efficace à une **distance modérée** et dans un **environnement relativement calme**. Cependant, certaines limitations ont été identifiées, notamment **la sensibilité aux interférences sonores et aux obstacles physiques**, qui peuvent altérer la réception des signaux.

- ❖ Des améliorations peuvent être envisagées pour rendre le système plus robuste et performant, comme l'**ajout d'un protocole bidirectionnel avec accusé de réception**, l'**optimisation du signal sonore via PWM**, ou encore l'**utilisation d'autres technologies de transmission comme l'infrarouge ou la radiofréquence**.