Architecture Temps Réel Pour L'embarqué
Driver RTDM pour un capteur Ultrason

Réaliser par :
SFAXI Nidhal
5A ESE 2024/2025

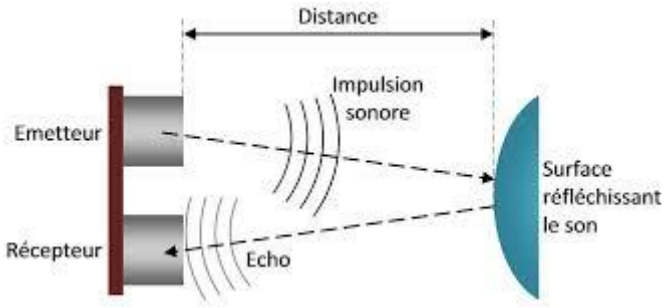
Table des matières

1 Introduction

Ce rapport a pour objectif de détailler, étape par étape, le processus d'implémentation d'un driver temps réel sous Xenomai, en utilisant un capteur ultrason comme exemple pratique. L'approche vise à maîtriser les techniques nécessaires au développement de drivers RTDM dans un environnement Linux temps réel. Bien que le capteur ultrason soit le cas d'étude principal, la démarche présentée peut être adaptée à d'autres types de capteurs avec seulement quelques ajustements spécifiques. Ce document se veut donc un guide pratique et structuré pour comprendre et reproduire cette méthodologie dans des projets similaires.
Les capteurs ultrasons sont largement utilisés dans les systèmes embarqués pour des applications telles que la mesure de distance, l'évitement d'obstacles et les systèmes de navigation autonome. Le principe repose sur la détection et la mesure du temps de vol des ondes sonores réfléchies par un objet.

2 Principe physique des capteurs ultrason

Les capteurs, quelle que soit leur nature, sont basés sur des principes physiques spécifiques. Ces principes sont détaillés dans la fiche technique du capteur et comprennent des informations cruciales sur le fonctionnement interne, les signaux d'entrée/sortie et les comportements temporels.
Dans le cas des capteurs ultrason, voici les étapes principales :



1. **Émission d'une onde sonore** :
   Le capteur émet un signal sonore de haute fréquence à l'aide de son émetteur (Trigger).
2. **Réflexion sur un obstacle** :
   L'onde rebondit sur un objet et retourne au capteur via le récepteur (Echo).
3. **Mesure du temps de vol** :
   Le temps entre l'émission et la réception est proportionnel à la distance parcourue.
4. **Conversion en distance** :
   En utilisant la vitesse du son dans l'air (343 m/s à 25°C), la distance est calculée avec la formule suivante :

$$Distance\ (cm) = \frac{Temps\ (\mu s) \times 343(m/s)}{20000}$$

3 Démarche d'implémentation

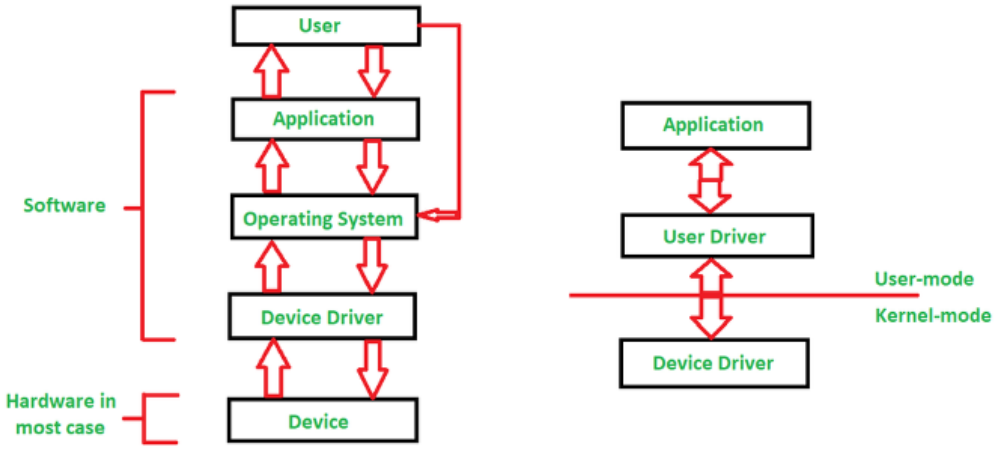


Image source : https://acervolima.com/driver-de-dispositivo-e-sua-finalidade/

3.1 Démarche générale pour implémenter un driver RTDM

L'implémentation d'un driver RTDM (Real-Time Driver Model) pour un capteur suit une méthodologie structurée et adaptable à différents types de capteurs. Cette démarche commence par l'étude approfondie des caractéristiques du capteur et se termine par la création d'une interface utilisateur via un fichier périphérique permettant la communication entre le Hardware et Software. Voici les étapes générales :

3.1.1 Étude de la fiche technique du capteur

- **Comprendre le principe physique** : Identifier le fonctionnement du capteur (émission/réception, types de signaux générés, principes physiques mis en jeu).

  La fiche technique fournit les détails nécessaires, tels que les protocoles de communication (I2C, SPI, GPIO, etc.), les tensions d'alimentation, et les caractéristiques des signaux (fréquence, amplitude, durée).

- **Analyser les équations spécifiques** : L'étude des équations ou des relations fournies dans la fiche technique permet de comprendre les calculs nécessaires pour interpréter les données brutes. Par exemple, pour un capteur ultrason, la relation entre le temps de vol et la distance est donnée par :

$$Distance\ (cm) = \frac{Temps\ (\mu s) \times 343\left(\frac{m}{s}\right)}{20000}$$

- **Identifier les fonctionnalités nécessaires** : Noter les fonctions spécifiques à implémenter, comme l'émission de signaux, la capture de données ou le traitement des interruptions.

3.1.2 Configuration des GPIO

- Configurer les broches GPIO en fonction des besoins du capteur. Cette étape peut inclure :
  - **Mode sortie** : Pour générer un signal d'émission (Trigger ou tout autre signal de commande).
  - **Mode entrée** : Pour lire les données renvoyées par le capteur (par exemple, signal Echo).
- Pour certains capteurs (comme l'ultrason), une broche GPIO unique peut être utilisée en mode alterné entrée/sortie. Cela simplifie le câblage mais nécessite une gestion précise via des délais et des reconfigurations.

3.1.3 Implémentation des fonctions de base liées au capteur

Chaque capteur nécessite des fonctions spécifiques basées sur son fonctionnement. Voici un exemple des fonctions de base à implémentés pour un capteur ultrason :

Émission d'une impulsion précise :

- Générer une impulsion haute de 10 µs pour activer le capteur. Utiliser des fonctions comme `udelay()` pour respecter la durée exacte.

Mesure du temps de vol :

- Capturer le signal renvoyé en détectant les **fronts montants** et **descendants**. Cela permet de mesurer la durée pendant laquelle le signal est haut, correspondant au temps de vol.
- Gérer les interruptions pour détecter les changements d'état du GPIO associé.

Calcul de distance :

- Convertir le temps mesuré en distance (ou toute autre donnée physique selon le capteur) à l'aide des équations définies dans la fiche technique.

3.1.4 Développement des fonctions RTDM

Pour intégrer le capteur dans un environnement temps réel, les étapes suivantes sont nécessaires :

Création d'un module RTDM :

- Définir les structures nécessaires pour le driver, comme `rtdm_driver` et `rtdm_device`.
- Configurer les opérations principales (`open`, `close`, `read`, etc.).

Gestion des interruptions et de la concurrence :

- Protéger les ressources critiques avec des verrous RTDM (`rtdm_lock_t`).

- Traiter les interruptions pour capturer les changements d'état du capteur en temps réel.

Fonctions utilitaires :

- Implémenter des fonctions pour gérer les délais, les configurations GPIO, et les calculs précis liés au capteur.

3.1.5 Création d'un fichier périphérique

- Attribuer un nom au périphérique (par exemple, `/dev/ultrasound_rtdm` pour un capteur ultrason).
- Permettre l'accès utilisateur aux données du capteur à travers des appels système standard (`open`, `read`, etc.).
  - Exemple : Une lecture simple retourne la distance mesurée ou une autre donnée pertinente.
  - Intégrer des mécanismes de gestion des erreurs si le capteur ne répond pas ou si les données sont incohérentes.

3.2 Application au cas d'un capteur ultrason

Dans le cas spécifique d'un capteur ultrason, la démarche suit les étapes décrites :

1. Configurer un GPIO unique en modes **sortie** (Trigger) et **entrée** (Echo).
2. Envoyer une impulsion haute de 10 µs pour activer le capteur.
3. Capturer le temps de vol en détectant les changements d'état du signal Echo.
4. Calculer la distance en fonction du temps mesuré et de la vitesse du son.
5. Offrir une interface utilisateur pour accéder aux mesures via un fichier périphérique.

4 Structure du Driver

4.1 Inclusion des bibliothèques:

<linux/module.h> : Fournit les macros nécessaires à la gestion des modules Linux (module_init, module_exit).
<linux/gpio.h> : Gestion des broches GPIO.
<linux/interrupt.h> : Gestion des interruptions pour capturer les changements d'état du GPIO Echo.
<linux/delay.h> : Fournit des fonctions de délai comme udelay pour générer des impulsions précises.
<rtdm/driver.h> : Interface RTDM pour développer un driver temps réel sous Xenomai.
<linux/ktime.h> : Fournit des horodatages avec une résolution élevée pour mesurer le temps de vol.

4.2 Définition des constantes et variables globales

GPIO_SIG : Broche GPIO utilisée pour interfacer le capteur.
DEVICE_NAME : Nom du périphérique pour identifier le driver.
TIMEOUT_US : Temps maximal pour attendre un retour Echo (1 seconde).
rtdm_lock_t lock : Verrou RTDM pour protéger la variable **distance_cm** lors des accès concurrents.

4.3 Implémentations des fonctions de base du driver

Nom de la fonction	Rôle principal
`pulse_in`	Mesure la durée d'un signal haut sur le GPIO avec timeout.
`send_pulse`	Génère une impulsion Trigger pour activer le capteur ultrason.
`measure_distance`	Calcule la distance d'un obstacle à partir du signal Echo et du temps de vol.
`ultrasound_read`	Permet de lire la distance mesurée via le fichier périphérique RTDM.
`ultrasound_open`	Ouvre le fichier périphérique (préparation des ressources).
`ultrasound_close`	Ferme le fichier périphérique (libération des ressources).
`ultrasound_init`	Initialise le module (configuration du GPIO, enregistrement du périphérique RTDM, etc.).
`ultrasound_exit`	Libère les ressources et désenregistre le module lors de son déchargement.

Voici une analyse complète des principales fonctions implémentées

4.3.1 Fonction simulant `pulseIn`

Structure:

```
static long pulse_in(int gpio, int state, long timeout)
```

```
{
    long start_time, end_time, duration;

    // Attente du changement d'état initial
    while (gpio_get_value(gpio) == state) {
        if (ktime_us_delta(ktime_get(), start_time) > timeout)
            return -1; // Timeout
    }

    // Capture du front montant
    start_time = ktime_to_us(ktime_get());
    while (gpio_get_value(gpio) != state) {
        if (ktime_us_delta(ktime_get(), start_time) > timeout)
            return -1; // Timeout
    }

    // Capture du front descendant
    end_time = ktime_to_us(ktime_get());
    duration = ktime_us_delta(end_time, start_time);
    return duration;
}
```

Rôle :

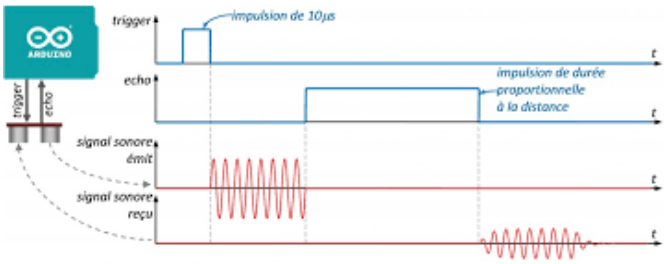
- **Entrées** :
  - `gpio` : Numéro du GPIO surveillé.
  - `state` : État (haut ou bas) à détecter.
  - `timeout` : Temps maximum d'attente (en microsecondes).
- **Sortie** :

  Retourne la durée du signal détecté (en microsecondes), ou `-1` en cas de dépassement du timeout.

- **Explications** :

  Cette fonction mesure la durée pendant laquelle un signal reste à l'état haut ou bas

4.3.2 Fonction d'envoi de la pulsation



Structure :

```
static void send_pulse(int gpio)
{
    gpio_direction_output(gpio, 1); // Configurer le GPIO en sortie
```

```
    udelay(10);                     // Maintenir l'état haut pendant
10 µs
    gpio_set_value(gpio, 0);        // Revenir à l'état bas
    udelay(2);                      // Pause pour la stabilité
    gpio_direction_input(gpio);     // Reconfigurer en mode entrée
}
```

Rôle :

- **Entrées** :
  - `gpio` : Numéro du GPIO pour envoyer la pulsation.
- **Explications** :
  - Envoie une impulsion précise de 10 µs pour déclencher le capteur ultrason.
  - Garantit la transition stable en mode entrée après l'impulsion.
  - Utilisation de `udelay` pour des délais précis nécessaires aux systèmes temps réel.

4.3.3 Fonction de mesure de distance

Structure

```
static int measure_distance(int trig_gpio, int echo_gpio)
{
    long duration;
    int distance;

    // Envoyer l'impulsion
    send_pulse(trig_gpio);

    // Mesurer la durée du signal Echo
    duration = pulse_in(echo_gpio, 1, 20000); // Timeout de 20 ms
    if (duration == -1)
        return -1; // Erreur si le timeout est dépassé

    // Calculer la distance (en cm)
    distance = (duration * 343) / 20000; // 343 m/s est la vitesse du
son
    return distance;
}
```

Rôle :

- **Entrées** :
  - `trig_gpio` : GPIO utilisé pour envoyer l'impulsion de déclenchement.
  - `echo_gpio` : GPIO utilisé pour lire le signal Echo.
- **Sortie** :
  - Retourne la distance mesurée (en cm), ou `-1` en cas d'erreur.
- **Explications** :
  - Combine `send_pulse` pour initier la mesure et `pulse_in` pour lire la réponse.
  - Convertit la durée mesurée en distance en tenant compte de la vitesse du son.

4.3.4 Initialisation du module

Structure :

```
static int __init ultrasound_init(void)

{
    int ret;

    // Initialiser le verrou RTDM
    rtdm_lock_init(&lock);

    // Configurer et demander accès au GPIO
    ret = gpio_request(GPIO_SIG, THIS_MODULE->name);
    if (ret != 0) {
        pr_err("Erreur : impossible de demander le GPIO %d\n",
GPIO_SIG);
        return ret;
    }

    gpio_direction_input(GPIO_SIG); // Configurer le GPIO comme
entrée

    // Enregistrer le périphérique RTDM
    ret = rtdm_dev_register(&ultrasound_device);
    if (ret != 0) {
        gpio_free(GPIO_SIG); // Libération du GPIO si échec
        pr_err("Erreur : enregistrement RTDM échoué\n");
        return ret;
    }

    pr_info("Module ultrason chargé\n");
    return 0;
}
```

Rôle :

- Initialise les ressources nécessaires au fonctionnement du module :
  - Verrou RTDM pour la gestion concurrente.
  - Configuration du GPIO.
  - Enregistrement du périphérique RTDM pour une utilisation par des applications utilisateur.
- Gère les erreurs en libérant les ressources en cas d'échec.

4.3.5 Déchargement du module

Structure

```
static void __exit ultrasound_exit(void)
{
    rtdm_dev_unregister(&ultrasound_device); // Désenregistrement
    gpio_free(GPIO_SIG);                     // Libération du GPIO
    pr_info("Module ultrason déchargé\n");
}
```

Rôle :

- Libère les ressources allouées pendant l'initialisation :
  - Désenregistrement du périphérique RTDM.
  - Libération du GPIO.
- Garantit une désinstallation propre du module.

4.3.6 Gestion du périphérique RTDM

Structure :

```
static struct rtdm_driver ultrasound_driver = {

    .profile_info = RTDM_PROFILE_INFO(ultrasound_rtdm,
RTDM_CLASS_TESTING, 1, 0),
    .device_flags = RTDM_NAMED_DEVICE,
    .device_count = 1,
    .ops = {
        .open = ultrasound_open,
        .close = ultrasound_close,
        .read_nrt = ultrasound_read,
    },
};

static struct rtdm_device ultrasound_device = {
    .driver = &ultrasound_driver,
    .label = DEVICE_NAME,
};
```

Rôle

- **`ultrasound_driver`** :
  Définit les comportements du module, y compris :
  - L'ouverture, fermeture et lecture du périphérique.
  - La classe et la version du profil.
- **`ultrasound_device`** :
  Associe un périphérique physique au pilote défini.

5 Tests et validation

Pour garantir le bon fonctionnement et la fiabilité du module RTDM, une série de tests a été effectuée. Ces tests permettent de valider à la fois les aspects matériels (capteur, GPIO) et logiciels (driver, interaction avec RTDM). Voici la démarche détaillée :

5.1 Branchement du Capteur

Vcc : Alimentation 5V.
GND : Masse.
SIG : I/O (`GPIO_SIG = 16`).

5.2 Compilation et insertion du module

Compilation du module :

Utiliser un Makefile adapté pour générer le module.

```
> nano Makefile
> make
```

Insertion du module :

Charger le module dans le noyau Linux.

```
> sudo insmod ultrasound_rtdm.ko
```

Vérification du chargement :

Confirmer que le module est chargé en listant les modules actifs :

```
> lsmod | grep ultrasound_rtdm
```

Vérification des logs :

Lire les messages de journalisation pour s'assurer que le module a été chargé correctement :

```
> dmesg | tail
```

5.3 Test fonctionnel du module

Ouvrir le périphérique RTDM :

Une application utilisateur peut interagir avec le driver via le périphérique `/dev/ultrasound_rtdm`.

Exemple de commande pour vérifier la présence du périphérique :

```
> ls /dev/ | grep ultrasound_rtdm
```

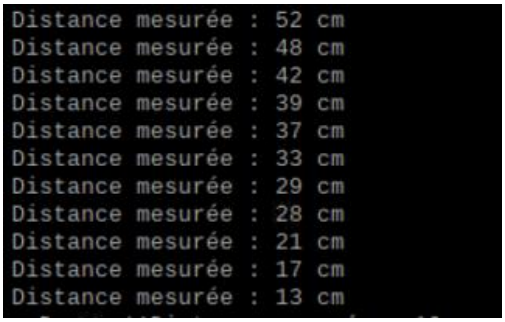
Lecture de la distance mesurée :

Un programme utilisateur peut lire la distance mesurée. Exemple de code en C pour tester :

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int fd = open("/dev/ultrasound_rtdm", O_RDONLY);
    if (fd < 0) {
        perror("Erreur ouverture du périphérique");
        return -1;
    }

    long distance;
```

```
        if (read(fd, &distance, sizeof(long)) < 0) {
            perror("Erreur lecture de distance");
            close(fd);
            return -1;
        }

        printf("Distance mesurée : %ld cm\n", distance);
        close(fd);
        return 0;
    }
```

Tester des scénarios :
Placez des obstacles à des distances connues (par exemple, 10 cm, 50 cm, 100 cm).



5.4 Nettoyage

Après les tests :
Décharger le module :

```
> sudo rmmod ultrasound_rtdm
```

Libérer les ressources GPIO si elles ne l'ont pas été automatiquement.

6 Conclusion

L'implémentation d'un driver temps réel pour un capteur ultrason sous Xenomai a permis de mettre en lumière les étapes essentielles pour développer un tel module : configuration des GPIO, gestion des ressources matérielles et interaction avec le noyau temps réel. Cette démarche constitue une base solide pour interfacer d'autres capteurs ou périphériques en environnement temps réel, et offre des perspectives pour des projets complexes nécessitant des performances critiques. Ce travail contribue ainsi à la maîtrise des techniques de développement temps réel, en les rendant applicables à un large éventail de dispositifs embarqués.

7 Ressources

Ressources sur le noyau Xenomai

1. **Documentation Officielle de Xenomai**
   - Explications détaillées sur l'installation et l'utilisation de Xenomai.
     Lien : https://xenomai.org/
2. **Compilation et patch du noyau Linux avec Xenomai pour Raspberry Pi**
   - Guide complet pour configurer et compiler un noyau Xenomai sur Raspberry Pi.
     Lien : https://github.com/MasterJoon/RPI-Xenomai.
3. **Exemple de gestion de PWM précis avec un noyau Xenomai**
   - Article et discussion sur l'utilisation de Xenomai pour générer des signaux PWM précis.
     Lien : https://community.robotshop.com/.

Ressources sur les drivers temps réel (RTDM)

1. **Introduction à l'interface RTDM**
   - Documentation officielle sur l'interface RTDM de Xenomai.
     Lien : https://xenomai.org/documentation/
2. **Guide pour développer des drivers temps réel pour Raspberry Pi**
   - Tutoriels pratiques pour créer des drivers RTDM en utilisant une plateforme Raspberry Pi.
     Lien : https://www.linuxtopia.org/

Ressources sur la configuration Raspberry Pi

1. **Configuration et optimisation du noyau Raspberry Pi pour temps réel**
   - Guide pour configurer et compiler le noyau Raspberry Pi avec les correctifs Xenomai.
     Lien : https://github.com/MasterJoon/RPI-Xenomai.
2. **Documentation officielle Raspberry Pi**
   - Références pour le matériel et les outils de développement Raspberry Pi.
     Lien : https://www.raspberrypi.org/documentation/

Ressources sur le kit Grove

1. **Documentation officielle de Grove (Seeed Studio)**
   - Informations et documentation pour les modules Grove (capteurs, buzzers, etc.).
     Lien : https://wiki.seeedstudio.com/Grove/
2. **Exemples et tutoriels pour capteurs Grove sur Raspberry Pi**
   - Guides pour utiliser les modules Grove sur Raspberry Pi.
     Lien : https://wiki.seeedstudio.com/Category:Grove/