Bureau d'Études Systèmes Embarqués et Temps Réels

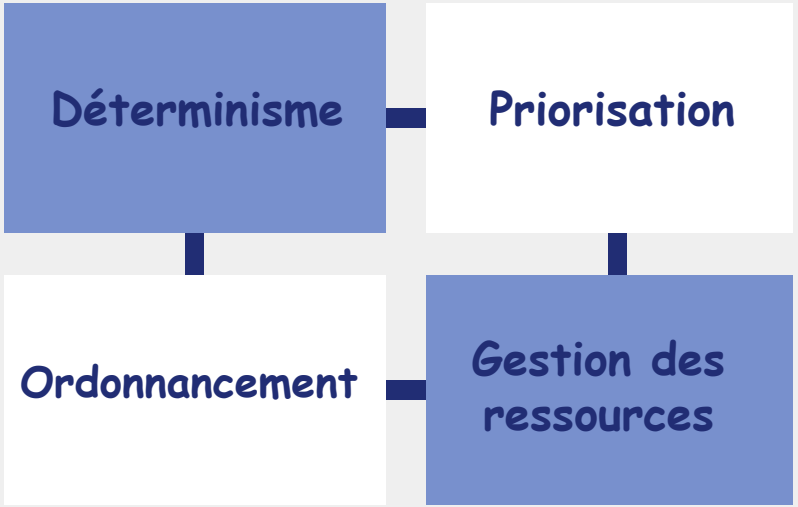
Elaboré par:
Cyrine LABIDI
Amal DJEMAI

Année Universitaire 2023-2024

Introduction

Les systèmes temps réels jouent un rôle vital dans divers secteurs industriels, assurant une performance, une sécurité et une fiabilité optimales. Leur fonctionnement repose sur des principes clés comme le déterminisme, la priorisation des tâches, l'ordonnancement et la gestion efficace des ressources disponibles.

Principes fondamentaux des systèmes temps réels



- **Déterminisme**

Les systèmes temps réels nécessitent une assurance de délais de réponse prévisibles. Cela implique une maîtrise et une limitation des délais de traitement, d'accès aux ressources et de communication.

- **Priorisation**

Les tâches et processus dans un système temps réel sont hiérarchisés en fonction de leur importance et de leurs contraintes temporelles. Ceci assure l'exécution prioritaire des opérations critiques.

- **Ordonnancement**

La planification des tâches est cruciale pour assurer le respect des contraintes temporelles. Divers algorithmes d'ordonnancement, comme ceux dédiés au temps réel, sont employés pour organiser la séquence d'exécution des tâches.

- **Gestion des ressources**

Les systèmes temps réels doivent efficacement administrer les ressources matérielles et logicielles afin d'assurer l'accès opportun des tâches aux ressources requises.

Défis des systèmes temps réels

- **Complexité**

La complexité des systèmes temps réels peut croître rapidement en raison de la diversité des tâches, de leurs interrelations et des contraintes temporelles qui leur sont associées.

- **Garantie de temps de réponse**

Assurer que toutes les opérations critiques sont réalisées dans les délais peut poser des défis, surtout dans des environnements complexes et changeants.a

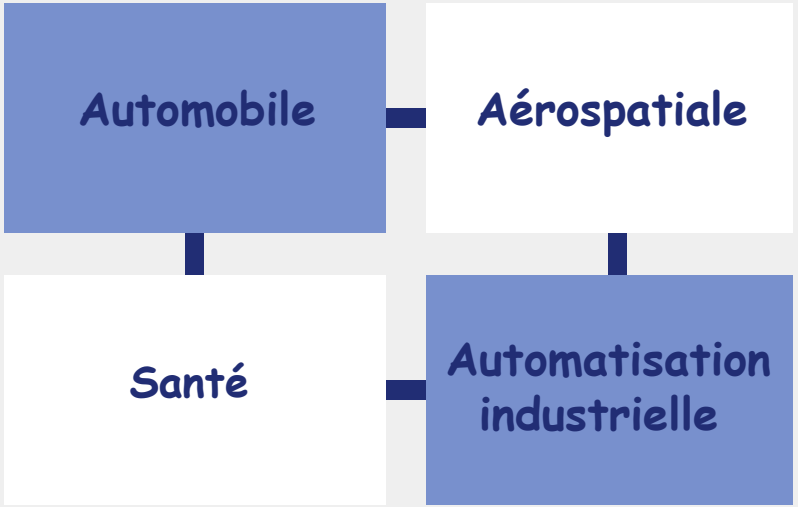
- **Gestion de la charge de travail**

Les systèmes temps réels doivent efficacement gérer les fluctuations de charge de travail, assurant que les tâches cruciales ne sont pas retardées en cas de pic d'activité.

- **Fiabilité et tolérance aux pannes**

Les systèmes temps réels doivent être conçus pour résister aux pannes matérielles ou logicielles, car une défaillance peut engendrer des conséquences significatives.

Applications des systèmes temps réels



- **Automobile**

Les systèmes de gestion moteur, de sécurité, de navigation et de divertissement des véhicules contemporains intègrent des systèmes temps réel afin d'assurer leur sécurité et leurs performances.

- **Santé**

Les équipements médicaux comme les moniteurs cardiaques et les pompes à insuline intègrent des systèmes temps réel pour assurer la sécurité et le bien-être des patients.

- **Aérospatiale**

Les avions, les satellites et les engins spatiaux s'appuient sur des systèmes temps réel pour des fonctions cruciales telles que la navigation, la communication et le contrôle.

- **Automatisation industrielle**

Les systèmes de contrôle des usines et des robots industriels s'appuient sur des systèmes temps réel pour garantir à la fois la production et la sécurité.

Évolution et tendances

- **Intelligence artificielle**

L'intégration de l'intelligence artificielle dans les systèmes temps réel pour améliorer la prise de décision et l'adaptabilité en temps réel.

- **Flexibilité et Scalabilité**

Des architectures plus flexibles et évolutives pour s'adapter aux besoins changeants et à l'expansion des systèmes temps réel.

- **Sécurité renforcée**

Une attention accrue à la cybersécurité pour prévenir les cyberattaques, notamment avec l'adoption de normes de sécurité et de protocoles plus robustes.

- **Automatisation avancée**

Le développement de techniques d'automatisation plus sophistiquées pour permettre aux systèmes temps réel de s'adapter rapidement à des environnements changeants.

Implémentation d'un système temps réel sur BeagleBone Black

Caractéristiques principales

- **Processeur** : Texas Instruments Sitara AM3358 base sur l'architecture ARM Cortex-A8 cadencé a 1 GHz
- **Mémoire** : 512 Mo de mémoire RAM DDR3
- **Stockage** : 4 Go de mémoire eMMC intégrée et emplacement pour carte microSD
- **Connectivité** : Port Ethernet 10/100, ports USB hôte et esclave.
- **Interfaces vidéo** : Sortie HDMI
- **Interfaces d'extension** : Deux en-têtes d'extension pour capes et accessoires
- **Système d'exploitation** : Prise en charge de divers systèmes d'exploitation Linux (Debian, Ubuntu, etc.)

Communauté active

- La BeagleBone Black compte sur une communauté active d'utilisateurs, présente sur de nombreux forums en ligne où ces derniers partagent des projets, des tutoriels et des astuces.

Capes et extensions

- La BeagleBone Black est compatible avec diverses "capes", des modules d'extension matériels qui se connectent aux en-têtes d'extension afin d'ajouter des fonctionnalités supplémentaires.

Applications courantes

- La BeagleBone Black est impliquée dans de multiples applications telles que la robotique, l'automatisation industrielle, les projets IoT, la domotique, l'informatique embarquée, l'éducation, et bien d'autres domaines.

Prix abordable

- La BeagleBone Black propose un excellent rapport qualité-prix par rapport à d'autres cartes de développement similaires, ce qui en fait une option abordable pour les projets électroniques.

BeagleBone Black Pinout

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPI0_CS0	17	18	SPI0_D1
I2C2_SCL	19	20	I2C2_SDA
SPI0_D0	21	22	SPI0_SCLK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SPI1_CS0
SPI1_D0	29	30	GPIO_112
SPI1_SCLK	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPPWM0
DGND	43	44	DGND
DGND	45	46	DGND

LEGEND

- POWER/GROUND/RESET
- AVAILABLE DIGITAL
- AVAILABLE PWM
- SHARED I2C BUS
- RECONFIGURABLE DIGITAL
- ANALOG INPUTS (1.8V)

P8

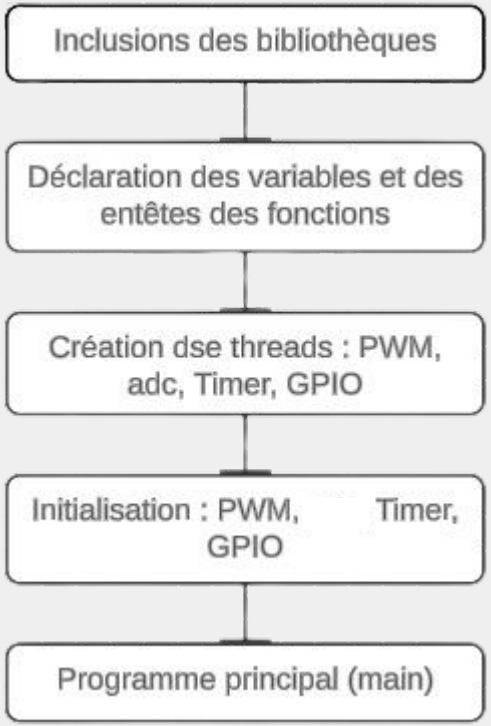
DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DAT0	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

Configuration de l'environnement de développement pour la BeagleBone Black

- **PREEMPT-RT** : La BeagleBone Black est compatible avec diverses distributions Linux, dont PREEMPT-RT. Voici les étapes pour la configurer :
  1. Télécharger la version de Debian conçue pour la BeagleBone Black depuis le site officiel de BeagleBoard.
  2. Flasher l'image PREEMPT-RT sur une carte microSD, puis insérer-la dans la BeagleBone Black.
  3. Démarrer la BeagleBone Black avec la carte microSD et connecter à la carte via SSH.
- **IDE Geany** : Nous avons employé Geany, un éditeur de texte léger et simple, adapté au développement sur la BeagleBone Black.
- **Cross-compilateur ARM** : Un compilateur croisé ARM est essentiel pour compiler le code source sur l'ordinateur de développement puis le transférer vers la BeagleBone Black.

Code et Réalisation

L'organigramme suivant représente la structure globale de notre programme :



Inclusion des bibliothèques :

Ces directives incluent différentes bibliothèques nécessaires au programme, telles que des bibliothèques pour les opérations système, la gestion des threads et la manipulation des GPIO.

Les variables globales du programme et les fonctions auxiliaires :

- Déclaration des fonctions et des variables globales utilisées dans tout le programme.
- structures pour la gestion des signaux, des lignes GPIO, et des timers.
- Les variables globales incluent des compteurs (c et duty), des durées (T), et des conditions pour la synchronisation des threads.

```
int ret;
pthread_attr_t attr;
timer_t timerid;
struct sigevent sev;
struct itimerspec trigger;
struct gpiod_line *lineRed;
struct gpiod_chip *chip;

pthread_cond_t blink_led;
pthread_cond_t adc;
pthread_cond_t pwm;
pthread_mutex_t lock;
```

- **Définition de la fonction ‘kbhit()’ :**

Cette fonction permet de détecter si une touche du clavier a été pressée.

```
char kbhit(void);
```

- **Définition de la fonction ‘ Tstimer_thread’ :**

Cette fonction est exécutée à chaque déclenchement du timer.

```
void Tstimer_thread(union sigval sv);
```


- **Définition de la fonction toggle_GPIO() :**

Cette fonction bascule l'état de la GPIO.

```
void toggle_GPIO(void);
```

Initialisation des périphériques:

On a développé des fonctions pour initialiser le timer, GPIO et le PWM :

```
void init_timer(void);
void init_GPIO(void);
int rt_init(void);
void init_pwm(void);
```

Ces fonctions vont être appelé au niveau de la fonction main.

- **init_timer :** initialiser le timer
- **init_GPIO :** Ouverture des lignes GPIO et les configurer
- **rt_init :** initialiser l'environnement temps réel du programme
- **init_pwm :** initialiser le PWM en configurant le pinmux pour activer la sortie PWM sur un port, définir la période et le rapport cyclique.

Création des Threads :

Il existe principalement trois threads :
- thread_gpio, thread_adc et thread_pwm.

```
void *thread_gpio(void *);
void *thread_adc(void *);
void *thread_pwm(void *);
```

- **thread_led :** Gère le clignotement de la LED en basculant l'état de la GPIO.
- **thread_adc :** Lit la valeur du signal analogique (ADC) à partir d'un fichier système.
- **thread_pwm :** Génère un signal PWM en fonction de la valeur lue par le thread ADC.

Threads en temps réel :

- La fonction **rt_init() :**
- Initialise l'environnement temps réel.
- Utilise les fonctionnalités de la bibliothèque POSIX pour gérer les attributs des threads et l'ordonnancement temps réel.

Verrouiller la mémoire :

```
/* Lock memory */
if(mlockall(MCL_CURRENT|MCL_FUTURE) == -1) {
        printf("mlockall failed: %m\n");
        exit(-2);
}
```

Initialiser le pthread et définir la taille de pile spécifique :

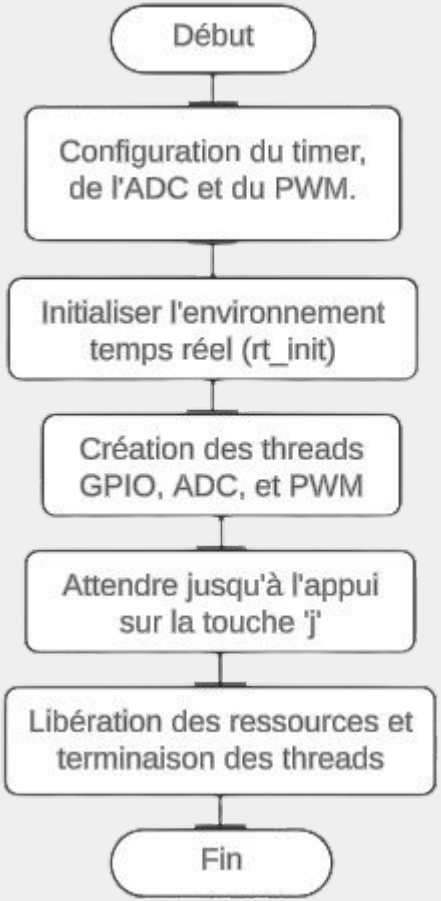
```
/* Initialize pthread attributes (default values) */
ret = pthread_attr_init(&attr);
if (ret) printf("init pthread attributes failed\n");

/* Set a specific stack size  */
ret = pthread_attr_setstacksize(&attr, PTHREAD_STACK_MIN);
if (ret) printf("pthread setstacksize failed\n");
```

Définir la politique de l'ordonnancement et la priorité :

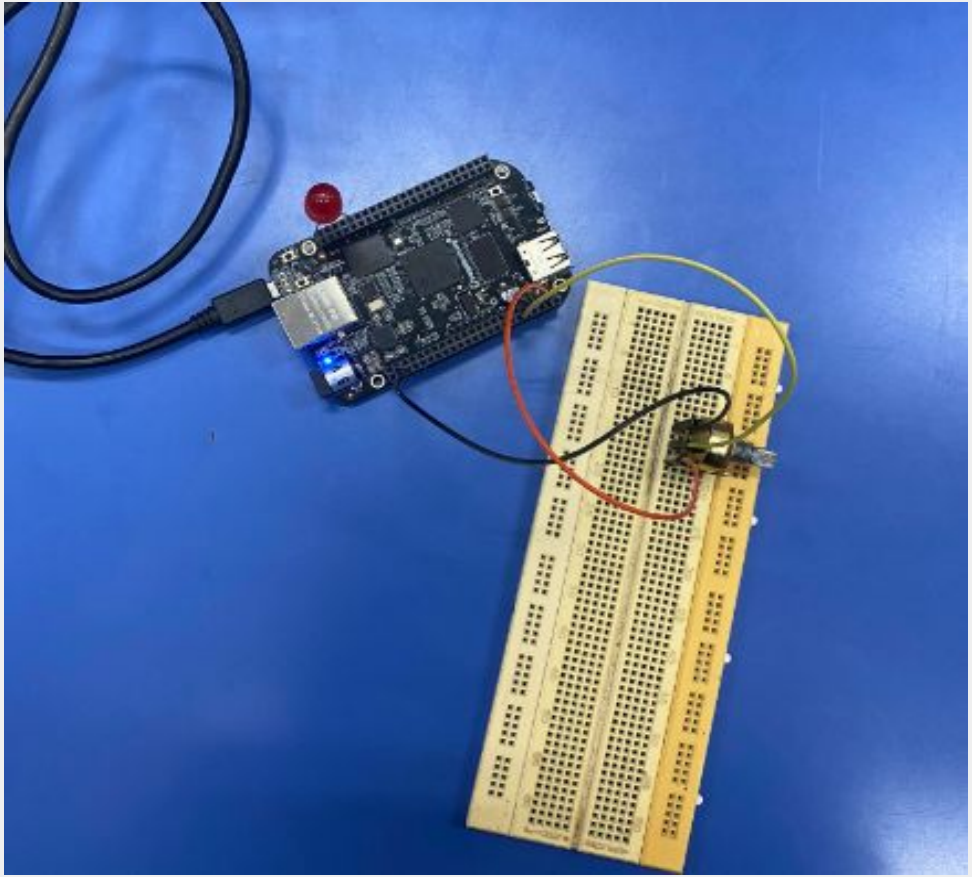
```
/* Set scheduler policy and priority of pthread */
ret = pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
if (ret) printf("pthread setschedpolicy failed\n");

param.sched_priority = 80;
ret = pthread_attr_setschedparam(&attr, &param);
if (ret) printf("pthread setschedparam failed\n");

/* Use scheduling parameters of attr */
ret = pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
if (ret) printf("pthread setinheritsched failed\n");
```

Programme principal (main) :

L’organigramme suivant montre le fonctionnement du programme principal



Test et Réalisation



Conclusion

Les systèmes temps réels jouent un rôle crucial dans de multiples secteurs industriels, assurant ainsi des niveaux optimaux de performance, sécurité et fiabilité. Leur fonctionnement repose sur des principes clés tels que le déterminisme, la priorisation, l'ordonnancement et la gestion des ressources. Bien que confrontés à des défis, les progrès technologiques récents renforcent la capacité des systèmes temps réels à répondre aux exigences toujours plus complexes de notre société interconnectée.

Merci pour votre attention