HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Reconfigurable & embedded
Digital Systems

Laboratoire 7

Gestion de surcharge

Départements : TIC
Unité d'enseignement PTR

Auteurs : **Rodrigo Lopez Dos Santos**
Urs Behrmann
Professeur : **Yorick Brunet**
Assistant : **Anthony I. Jaccard**
Classe : **PTR**
Salle de labo : **A09**
Date : **28.05.2025**

Contents

1 Introduction

Dans le cadre de l'unité d'enseignement PTR, ce laboratoire a pour objectif d'étudier la gestion de la surcharge dans un système temps réel embarqué. À travers différentes étapes pratiques, il s'agit d'observer le comportement du système sous forte charge, de mettre en place des mécanismes de détection de surcharge à l'aide d'un watchdog, puis d'implémenter des stratégies de dégradation fonctionnelle pour préserver la stabilité et la réactivité du système.

L'approche proposée consiste à simuler une surcharge progressive, à détecter automatiquement les situations critiques et à adapter dynamiquement le fonctionnement des tâches, notamment celle de traitement vidéo. Ce rapport détaille les observations réalisées, les choix d'implémentation et les résultats obtenus lors de chaque étape du laboratoire.

2 Architecure du projet

`code/` Contient l'ensemble des étapes de développement du code.

- `étape 1/` Première phase du développement.
- `étape 2/` Deuxième phase du développement.
- `étape 3/` Troisième phase du développement.

`doc/` Dossier réservé à la documentation (peut contenir rapports, schémas, etc.).

2.1 Compilation

Pour compiler le projet, il faut se placer à la racine de l'étape souhaitée (par exemple l'étape 1) :

```
/code/étape 1/
```

Ensuite, générer le fichier `Makefile` à partir du `CMakeLists.txt` en utilisant la commande suivante :

```
cmake -DCMAKE_TOOLCHAIN_FILE=/home/reds/Desktop/ptr/de1soc-sdk/share/...
    buildroot/toolchainfile.cmake .
```

Enfin, compiler le projet avec :

```
make
```

2.2 Fichiers ajoutés

Dans cette section, nous présentons les fichiers ajoutés au projet, en expliquant leur rôle fonctionnel sans entrer dans le détail du code.

`commun.h` Ce fichier d'en-tête centralise la configuration générale du système à travers des définitions de constantes partagées, telles que les périodes d'exécution des tâches temps réel (exprimées en nanosecondes), les priorités, et les seuils de dégradation du mode vidéo. Il définit également des types énumérés pour gérer les différents états de fonctionnement du système vidéo (mode normal, dégradé 1 et dégradé 2), ainsi que les types de réduction appliqués.

`canary.h` & `canary.c` Ces fichiers définissent et implémentent une tâche appelée « canary ». Elle agit comme une sonde système, s'exécutant à intervalle régulier pour incrémenter un compteur partagé. Cette tâche sert de référence temporelle que le système peut surveiller. Elle utilise un timer EVL et fonctionne avec des priorités temps réel. Son bon fonctionnement est essentiel pour que le système de surveillance (watchdog) puisse détecter d'éventuelles surcharges ou blocages.

`watchdog.h` & `watchdog.c` Ces fichiers définissent et implémentent la tâche watchdog, qui a pour rôle de surveiller la régularité des incréments du compteur du canary. Elle détecte les retards ou absences d'incréments, signe de surcharge ou de dysfonctionnement. En fonction de la gravité de la situation, elle ajuste dynamiquement le mode vidéo du système en passant par des états de dégradation (normal → dégradé 1 → dégradé 2) ou en revenant à un état plus stable si les conditions s'améliorent. En cas d'erreur grave persistante, elle peut arrêter le système pour éviter des conséquences critiques.

Afin de prendre en compte les fichiers ajoutés lors de la compilation, il est impératif de modifier la ligne de construction dans le `CMakeLists.txt` comme suit :

```
add_executable(main main.c video_setup.c load_setup.c canary.c watchdog.c)
```

3 Etape 1 : Observation de la surcharge

3.1 Détection des overruns

Pour détecter les overruns (dépassements de période) dans la tâche vidéo, nous utilisons le compteur de ticks retourné par le timer EVL à chaque réveil de la tâche. Si la valeur de `ticks` est supérieure à 1, cela signifie que la tâche n'a pas pu traiter toutes les images dans le temps imparti et qu'une ou plusieurs périodes ont été manquées. Un message d'avertissement est alors affiché pour chaque overrun détecté.

3.2 Modification du chargement vidéo

Initialement, le fichier vidéo était ouvert à chaque début de boucle principale, ce qui introduisait une surcharge inutile et des accès disques fréquents. Nous avons modifié le code pour ouvrir le fichier une seule fois au début de la tâche vidéo, puis réinitialiser le pointeur de fichier à la position zéro (`fseek(file, 0, SEEK_SET)`) à chaque nouvelle boucle sur les images. Cette optimisation réduit la charge système et améliore la stabilité temporelle de la tâche vidéo.

3.3 Observation

Lors de nos essais, nous avons constaté que les premiers overruns apparaissent dès que la charge atteint 36. À ce stade, le système reste globalement stable, mais des dépassements de période sont régulièrement signalés dans la console. Cela indique que la tâche vidéo n'a plus toujours le temps de traiter chaque image dans la fenêtre temporelle prévue, ce qui se traduit par une perte de fluidité et un risque de dégradation progressive des performances si la charge continue d'augmenter.

4 Etape 2 : Détection de la surcharge avec watchdog

4.1 Paramètres des tâches

Tâche	Période	Fréquence	Priorité
Vidéo	66.67 ms	15 Hz	50
Load	100 ms	10 Hz	70
Canari	10 ms	100 Hz	40
Watchdog	100 ms	10 Hz	90

4.2 Choix des priorités

- **Load** > **Vidéo** : La tâche Load simule une surcharge progressive en fonction des commutations (*switches*). Elle est volontairement prioritaire par rapport à Vidéo, afin que cette dernière soit affectée en cas de surcharge.
- **Load** > **Vidéo** > **Canari** : En cas de surcharge, la tâche Canari (dont le rôle est de s'exécuter très fréquemment) n'arrive plus à incrémenter son compteur. Cette baisse d'activité permet de détecter la surcharge via le Watchdog qui surveille ce compteur. D'où l'importance que Canari ait une priorité plus faible.
- **Watchdog** > **Load** > **Vidéo** > **Canari** : La tâche Watchdog doit toujours pouvoir s'exécuter, même en cas de surcharge, pour surveiller le bon fonctionnement du Canari. Elle possède donc la priorité la plus élevée. Cela garantit que le système puisse détecter une surcharge critique et éventuellement déclencher un arrêt de sécurité.

4.3 Choix des fréquences

- **Canari** ≫ **Watchdog** : Canari s'exécute 10 fois plus souvent que Watchdog. Cela permet au Watchdog de vérifier que le compteur du Canari a bien été incrémenté d'au moins 10 entre deux vérifications. Si ce n'est pas le cas, cela indique une surcharge.
- **Load** = **Watchdog** : Même fréquence mais rôles opposés : Load génère la surcharge, Watchdog la détecte. Leur fréquence identique garantit une surveillance cohérente.
- **Canari** ≫ **Vidéo** : Avec une période très courte (10 ms), Canari détecte rapidement toute baisse de performance. Son exécution fréquente est essentielle pour assurer la réactivité du mécanisme de détection.

5 Etape 3 : Fonctionnalité dégradée

Modes de compensation implémentés

Trois modes de compensation peuvent être sélectionnés dynamiquement via les touches du DE1-SoC :

- **Aucune compensation (MODE_NONE)** : Traitement complet, sans réduction de charge.
- **Réduction d'échelle (MODE_REDUCTION_SCALE)** : Traitement sur une image réduite (par facteur 2 ou 4), puis réagrandie.
- **Réduction de complexité (MODE_REDUCTION_COMPLEXITY)** : Simplification du traitement appliqué à chaque image.

5.1 Traitement vidéo selon le mode de compensation et le niveau de dégradation

Mode de compensation	**Niveau vidéo**	**Traitement appliqué**
Aucune (NONE)	Tous	Gris 8 bits, convolution, RGBA
Réduction d'échelle (SCALE)	NORMAL	Idem NONE
	DEGRADED_1	Réduction ×2, gris, convolution, agrandissement ×2
	DEGRADED_2	Réduction ×4, gris, convolution, agrandissement ×4
Réduction complexité (COMPLEXITY)	NORMAL	Gris 8 bits, convolution, RGBA
	DEGRADED_1	Gris 32 bits (sans convolution)
	DEGRADED_2	Affichage brut (aucun traitement)

5.2 Sélection du mode de compensation

Le mode de compensation peut être changé à la volée via les touches du DE1-SoC :

- Touche 0 : Aucune compensation
- Touche 1 : Réduction d'échelle
- Touche 2 : Réduction de complexité

5.3 Stratégie de gestion de la surcharge

Le système utilise la différence de fréquence entre la tâche Canari (100 Hz) et le Watchdog (10 Hz) pour détecter la surcharge. À chaque période du Watchdog (100 ms), il vérifie que le compteur du Canari a bien été incrémenté d'au moins 10. Le retard est mesuré et intégré dans une moyenne mobile pour lisser les variations.

Trois seuils sont définis pour adapter dynamiquement le comportement de la tâche vidéo :

- Si la moyenne mobile du retard est > `VIDEO_MODE_NORMAL_TRESHOLD` (1) : passage immédiat en `VIDEO_MODE_DEGRADED_1`
- Si la moyenne mobile du retard est > `VIDEO_MODE_DEGRADED_1_TRESHOLD` (2) : passage immédiat en `VIDEO_MODE_DEGRADED_2`
- Si la moyenne mobile du retard est > `VIDEO_MODE_DEGRADED_2_TRESHOLD` (9) : arrêt du système

Les retours à un mode moins dégradé ne sont possibles qu'après plusieurs cycles consécutifs sous le seuil correspondant :

- Retour à `VIDEO_MODE_NORMAL` après 5 cycles sous le seuil 1
- Retour à `VIDEO_MODE_DEGRADED_1` après 15 cycles sous le seuil 2

Le Watchdog écrit le mode vidéo courant dans une variable atomique. La tâche vidéo lit ce mode et adapte son exécution selon le mode de compensation sélectionné. Cette logique évite les oscillations rapides entre les modes et assure une adaptation progressive à la charge réelle du système.

5.4 Résumé du fonctionnement

- La surcharge est détectée automatiquement par le Watchdog via une moyenne mobile du retard du Canari.
- Le mode vidéo (NORMAL, DEGRADED_1, DEGRADED_2) est ajusté dynamiquement selon les seuils définis.
- Les transitions vers un mode moins dégradé nécessitent plusieurs cycles consécutifs sous le seuil.
- L'utilisateur peut choisir à tout moment la stratégie de compensation (aucune, réduction d'échelle, réduction de complexité).
- Le traitement vidéo s'adapte en temps réel pour préserver la stabilité du système.

5.5 Seuils d'apparition des overruns selon le mode

Mode de compensation	**Charge à l'apparition des overruns**
Aucun (NONE)	33
Réduction d'échelle (SCALE)	65
Réduction de complexité (COMPLEXITY)	76

6 Conclusion

Ce laboratoire nous a permis d'aborder concrètement la problématique de la surcharge dans un système temps réel embarqué. Nous avons d'abord observé l'apparition des overruns et identifié les limites du système en conditions nominales. L'intégration d'un watchdog et d'une tâche canari a permis de détecter automatiquement les situations de surcharge, puis d'adapter dynamiquement le traitement vidéo grâce à différents modes de compensation.

Les résultats montrent que la réduction de la charge de traitement, que ce soit par simplification des algorithmes ou par réduction de la résolution, permet d'augmenter significativement la robustesse du système face à la surcharge. On observe ainsi que le seuil d'apparition des overruns peut être doublé grâce à ces méthodes, passant d'environ 33 à plus de 65, voire 76 selon le mode choisi.

De plus, la gestion dynamique des modes, associée à une logique de récupération progressive avec compteur dans le watchdog, a permis d'éliminer complètement les oscillations entre deux modes d'affichage. Cette approche garantit une stabilité accrue tout en maintenant la meilleure qualité de service possible selon les ressources disponibles.

Ce travail met en évidence l'importance de la surveillance et de l'adaptation dans la conception de systèmes embarqués fiables et réactifs.