INF3610 – Systèmes embarqués

Automne 2024

TP No. 1 (Partie 2)

Groupe 02

[redacted]

Soumis à : [redacted]

Vendredi 25 Octobre 2024

Questions pour le rapport

Question 1 Il est fortement recommandé d'utiliser l'option OS_OPT_POST_NO_SCHED dans l'appel des fonctions gpio_isr0, gpio_isr1 et fit_timer_isr0. Expliquez la raison.

L'option `OS_OPT_POST_NO_SCHED` est fortement recommandée pour les appels aux fonctions `gpio_isr0`, `gpio_isr1` et `fit_timer_isr0`, car elle évite une réorganisation immédiate de l'ordonnanceur après l'envoi d'un signal de poste via `OSFlagPost`. Cette approche est essentielle dans le contexte des interruptions (ISR) pour plusieurs raisons.

Tout d'abord, les gestionnaires d'interruption doivent être exécutés rapidement. Si une réorganisation des tâches était effectuée immédiatement, cela introduirait des délais qui pourraient nuire à la réactivité du système. En empêchant cette réorganisation, l'option `NO_SCHED` garantit que l'ISR peut s'exécuter de manière prioritaire, ce qui est crucial pour répondre rapidement aux événements externes.

De plus, cette option permet de conserver l'ordre des interruptions. Elle assure que les ISR sont traitées dans l'ordre d'arrivée, sans interruption par d'autres tâches, ce qui est vital pour le bon fonctionnement d'applications en temps réel. Enfin, en réduisant les changements d'état et en optimisant l'utilisation des ressources système, l'option `OS_OPT_POST_NO_SCHED` contribue à une gestion plus efficace des ressources, permettant ainsi un fonctionnement fluide et performant du système.

Question 2 Décrivez le rôle des fonctions suivantes dans la tâche StartupTask :

• *initialize_gpio0();*

• *initialize_gpio1();*

Ces deux fonctions initialisent les GPIOs pour la gestion des interruptions matérielles. Elles configurent les broches spécifiques pour permettre au système de détecter et répondre aux événements d'entrée/sortie sur le matériel. Cela inclut la configuration des registres pour la gestion des interruptions. Ces initialisations sont cruciales car elles permettent à votre système d'interagir avec les périphériques externes via les interruptions.

• *initialize_axi_intc();*

Cette fonction initialise le contrôleur d'interruptions AXI (AXI Interrupt Controller). L'AXI est une architecture standard pour la gestion des interruptions dans les systèmes embarqués. Cette étape permet de configurer le contrôleur d'interruptions pour qu'il puisse gérer correctement les interruptions provenant de différents périphériques (comme les GPIOs) et les acheminer vers le processeur.

• *connect_axi();*

Cette fonction lie les interruptions spécifiques du matériel (comme celles générées par les GPIOs ou d'autres périphériques) au contrôleur d'interruptions AXI. Elle assure que chaque interruption matérielle est correctement dirigée vers le contrôleur pour que le système d'exploitation puisse y répondre.

<u>Question 3</u> Concernant la manipulation 4,

a. Expliquez la différence entre une interruption privée FIQ et IRQ (celles que vous avez utilisée dans ce lab).

Les interruptions FIQ (Fast Interrupt Request) et IRQ (Interrupt Request) sont deux types d'interruptions dans les systèmes embarqués. La principale différence réside dans leur priorité et leur rapidité d'exécution :

- **FIQ (Fast Interrupt Request)** : Ce type d'interruption est traité plus rapidement que les IRQ grâce à un vecteur d'interruption spécifique qui permet un traitement plus rapide des événements critiques. Les FIQ ont une priorité plus élevée et sont utilisées pour des événements nécessitant une réponse immédiate.
- **IRQ (Interrupt Request)** : Les IRQ sont des interruptions avec une priorité standard et sont souvent utilisées pour des événements qui, bien qu'importants, ne nécessitent pas une réponse aussi rapide que les FIQ. Les IRQ sont plus couramment utilisées pour des interruptions régulières.

b. Expliquez la différence entre une interruption IRQ privée et partagée.

Interruption IRQ privée : Une interruption privée est affectée à un seul processeur ou noyau. Cela signifie qu'une seule unité de traitement peut répondre à cette interruption. Ce type est généralement utilisé dans les systèmes embarqués pour des tâches spécifiques liées à un seul processeur.

Interruption IRQ partagée : Une interruption partagée peut être gérée par plusieurs processeurs ou unités. Cette approche est utilisée dans des systèmes multi-processeurs ou multi-cœurs, où plusieurs unités peuvent répondre à la même interruption, assurant ainsi une meilleure répartition de la charge de travail et une gestion plus flexible des interruptions.

En résumé, les interruptions privées, attribuées à un seul processeur, garantissent des réponses rapides et spécifiques, ce qui les rend idéales pour des tâches précises dans des environnements embarqués. En revanche, les interruptions partagées, qui permettent à plusieurs unités de traitement de répondre à la même interruption, favorisent une meilleure distribution de la charge de travail dans des systèmes multi-processeurs

c. Expliquez votre démarche pour la réalisation de la manipulation 4 (passage de l'interruption IRQ privée à partagée).

1. Modification de la configuration dans Vivado :

- Dans Vivado, la configuration du contrôleur d'interruptions (AXI Interrupt Controller) a été modifiée pour permettre la gestion des interruptions partagées.
- Cela a été fait en ajustant les paramètres du contrôleur, en activant la gestion des interruptions multiples et en ajoutant les sources d'interruption pertinentes (boutons, interrupteurs, timer) à la liste des périphériques gérés.
- Cette configuration permet au contrôleur de détecter et de gérer plusieurs interruptions via une seule ligne d'IRQ, ce qui est la base de la gestion des interruptions partagées.

2. Adaptation du code dans `interrupts.h` :

- La gestion des interruptions a été modifiée dans le fichier `interrupts.h` pour intégrer la logique des interruptions partagées.
- J'ai défini une ISR partagée appelée `shared_isr()` qui traite les interruptions générées par différents périphériques (GPIOs et timer) en une seule routine.
- L'ISR partagée identifie l'origine de l'interruption en utilisant le registre de statut du contrôleur d'interruptions (`XIntc_GetIntrStatus`) et déclenche ensuite l'ISR appropriée en fonction du type d'interruption (boutons, interrupteurs ou timer).
- Des macros et des fonctions spécifiques ont été ajoutées pour vérifier l'état des interruptions et pour réinitialiser le registre de statut après leur traitement.