Département de génie informatique et génie logiciel

INF3610

Systèmes embarqués

Laboratoire 2

Routeur sur puce FPGA

Soumis par

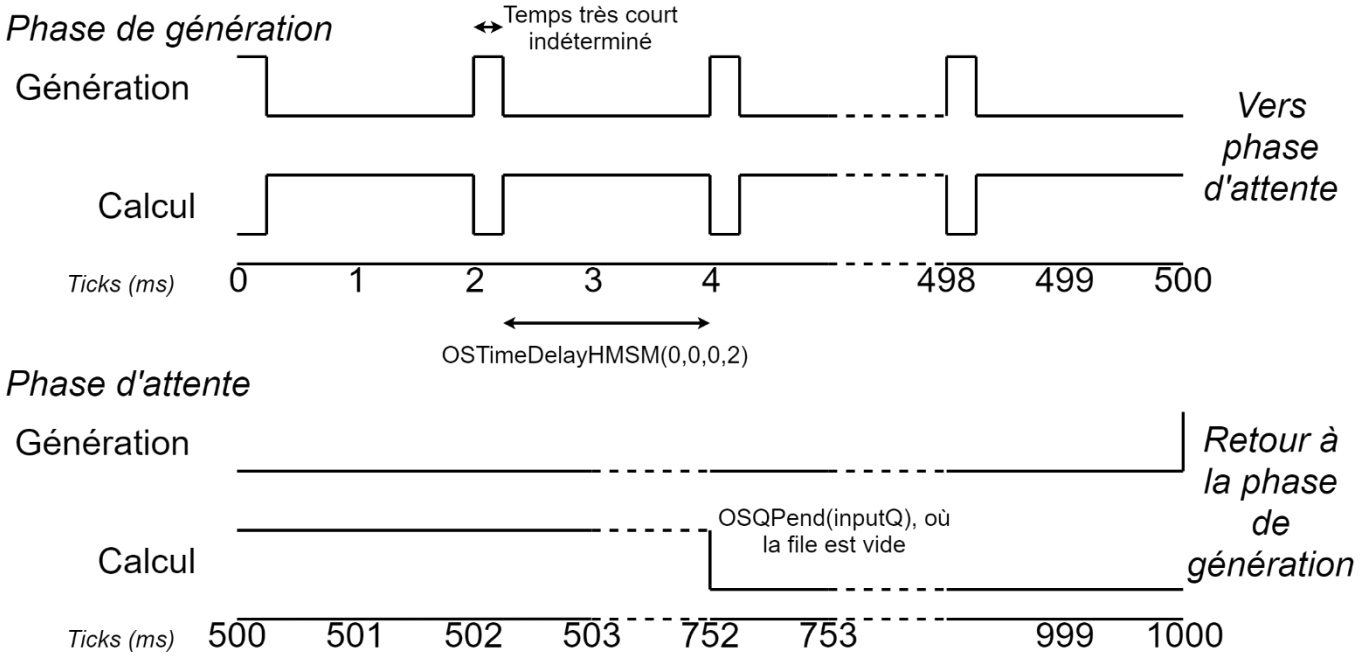
Félix Boulet, #1788287

Giuseppe La Barbera, #1799919

le 26 octobre 2017

Questions

1. Ordonnancement des tâches de génération et de calcul (traitement) des paquets



2. **A)** La phase de génération peut générer un maximum de 250 paquets, pour une durée de 500 ms au maximum ; en effet, la tâche appellera 250 fois un OSTimeDelay de 2 ms, équivalents à 2 ticks chacun car les ticks sont configurés pour durer 1 ms selon OS_TICKS_PER_SEC = 1000 dans os_cfg_r.h. Il est en théorie possible de donner ces 500 ms de temps de calcul à la tâche de calcul lors de la phase de génération (cela ne tient pas compte des changements de contexte ou des autres variations de délai, ni du petit temps de traitement pour la génération d'un paquet), donc il suffit de diviser ce temps par le temps de traitement d'un paquet. Nous avons vérifié expérimentalement que ce temps correspond à 3 ms (3 ticks) à l'aide de la fonction OSTimeGet(), donc il est possible de traiter $\frac{250}{3} = 166.\overline{6}$ paquets lors de la période de génération. Puisque les paquets doivent être traités en entier, on pourra arrondir ce nombre à la baisse à 166 paquets. Ensuite, puisque la phase d'attente ne génère pas de nouveaux paquets et correspond simplement à un délai de 500 ms, il ne reste que $250 - 166 = 84$ paquets, qui peuvent alors tous être traités lors de celle-ci.

B) Lorsque la phase de génération est terminée, il devrait rester au maximum 84 paquets à traiter dans la file entre la tâche génération et calcul, tel que démontré en A). La file devrait donc avoir cette taille au minimum, idéalement multipliée par deux (donc 168) pour prendre en compte les délais des changements de contexte et les délais d'interruptions qui ne sont pas constants. Lorsque testé expérimentalement dans le programme, une taille de 84 semble fonctionner relativement bien, toutefois, il arrive parfois qu'environ 5 ou 6 paquets soient rejetés à l'entrée de la FIFO, ce qui concorde avec notre calcul théorique puisque celui-ci ne tient pas compte des variations de délai et des changements de contexte.

3. *Durant la phase de génération :*
   Le temps d'utilisation du CPU devrait être approximativement 100% puisque les deux tâches ne sont jamais simultanément bloquées ou en délai. Si l'on se fie à notre calcul précédent toutefois, et qu'on assume que $166/166.\overline{6} = 99.6\%$ des paquets théoriquement traitables sont réellement traités, on se retrouve plutôt avec cela comme pourcentage d'utilisation CPU.
   Durant la phase d'attente :
   Puisqu'il reste au maximum 84 paquets à traiter, et que chaque paquet est traité en 3 ms, le CPU aura un temps d'utilisation maximal de $\frac{3*84}{500} = 50{,}4\%$. Pour les 248 ms restantes, les deux tâches seront inactives (le calcul est bloqué en attente de la *queue* et la génération est en délai). Toutefois, si la tâche de génération produit un nombre inférieur de paquets (100 par exemple), le temps d'utilisation s'en retrouvera réduit (avec 100 paquets générés, il resterait 34 paquets dans la file, menant à un temps d'utilisation de 20,4%).

4. Oui, il est possible d'effectuer le traitement de ces tâches dans les *handlers* d'interruption. En effet, les *handlers* dans leur implémentation actuelle ne font que faire un *SemPost* du sémaphore associé à chaque tâche respective afin de la laisser effectuer son traitement (qui dépend d'un *SemPend* sur le sémaphore en question). Un avantage de cette méthode serait une gestion encore plus prioritaire (donc sans changement de contexte) des arrêts et réinitialisations du système, qui se font à l'aide des tâches *TaskStop* et *TaskResume*, déjà plus prioritaires que toutes les autres tâches. On s'assurerait donc d'une gestion complètement

asynchrone des interruptions (arrêt et redémarrage immédiats). Toutefois, c'est un désavantage dans le cas de la tâche de statistiques, qui s'afficherait alors en tout temps à toutes les trois secondes (elle est normalement moins prioritaire que toutes les autres tâches sauf le forwarding). De plus, les ISR ne peuvent acquérir ou libérer un mutex, donc il serait impossible de protéger l'impression via *xil_printf*, et on se retrouverait avec un ordre d'affichage potentiellement sens dessus dessous.

5. Selon la documentation du contrôleur d'interruptions AXI (https://www.xilinx.com/products/intellectual-property/axi_intc.html), une interruption matérielle qui surviendrait pendant le traitement d'une première sera immédiatement traitée (par préemption, donc un *nested interrupt*) si celle-ci est plus prioritaire que celle actuellement traitée. Pour déterminer la priorité d'une interruption matérielle, le contrôleur vérifie quel est le bit le moins significatif de la concaténation d'interruptions (une interruption arrivant sur le port 0 serait plus prioritaire qu'une interruption arrivant sur le port 1).