# Question 1

GPIO : GPIO est pour General Purpose Input/Ouput. Le fait de le relier aux switches permet de récupérer une interruption manuelle enclenchée par un utilisateur de la carte. Ainsi, comparément aux FIT, c’est nous ici qui enclenchons les interruptions grâce au GPIO. Nous pouvons donc relier cette interruption à notre fonction d’affichage des statistiques. De ce fait, en touchant aux switches, une interruption sera déclenchée et nous pourrons voir les statistiques en rapport aux files, terminaux etc.

FIT : FIT est pour Fixed Interval Timer. Ces modules nous permettent de créer des minuteries à intervalle fixe et ils vont donc envoyer un signal d’interruption à chaque nombre x de cycles (choisie par nous). Ces interruptions seront utilisées pour répéter une routine à chaque intervalle de temps. Nous attacherons ces interruptions à deux de nos fonctions ISR. L’interruption à chaque seconde sera reliée à notre fonction de génération alors que celle au 3 secondes est attachée à notre fonction de vérification.

Contrôleur AXI : Ce contrôleur sert d’intermédiaire entre nos différentes sources d’interruptions et le cœur du processeur ARM. Il va donc être utilisé pour multiplexer les interruptions à l’entrée d’interruptions du cœur. C’est donc lui qui va indiquer au CPU quelles sont les interruptions qui sont activées.

# Question 2

Le Zynq-7020 possède 195 ports I/O multistandards de 3.3V. Ceux-ci pourraient techniquement tous être connectés à des périphériques. Dans notre cas, le contrôleur d’interruptions AXI reçoit en entrée les 8 switches du GPIO, en plus des deux FIT que nous avons défini. Ces 8 switches occupent une des interruptions qu’il peut gérer, tout comme les FIT. Le contrôleur AXI peut gérer un total de 16 interruptions, ce qui correspond à la partie de logique programmable des interruptions de périphériques partagés. Cela fait donc un total de 195 + 8 + 16 – 1 = 218 périphériques.

# Question 3

À toutes les secondes, fit timer 0 génère une interruption, que le contrôleur d’interruption AXI achemine à µC-OS II. Lorsque l’interruption est reçue par le système, ce dernier s’occupe tout d’abord désactiver les interruptions. Puis, il sauvegarde le contexte, en sauvegardant tous les registres sur la pile SVC.

Ensuite, si le système est complètement initialisé (OSRunning == 1), il incrémente la variable mémorisant le nombre d’interruptions imbriquées (OSIntNesting). Sinon, il lance directement une requête d’interruption (IRQ).

Dans le cas où le système est bel et bien initialisé, deux scénarios sont possibles : une tâche est interrompue (lorsque OSIntNesting est égal à un après incrémentation), ou une requête d’interruption est interrompue. Dans le premier scénario, le pointeur de pile SP est assigné à la pile de la tâche, alors que dans le deuxième, on utilise la pile d’exceptions. Finalement, dans les deux scénarios, une requête d’interruption est lancée.

La requête d’interruption est alors traitée par le handler définit dans bsp\_init.c, qui appelle la routine d’interruption (ISR) que nous avons définie dans simAvion.c.