Nom : AIDER
Prénom : Smail
N°Etudiant : 3603379
Parcours : SAR
Responsable : M Alain Greiner

Compte-Rendu TP7
Contrôleur DMA

B) *Contrôleur DMA*

Question B1

Les registres adressables du contrôleur DMA:
- SOURCE @(0x00) ; R/W :
→ L'accès en écriture est ignoré si l'automate DMA n'est pas dans l'état IDLE.
- DEST @(0x04) ; R/W :
→ L'accès en écriture : idem que SOURCE.
- LENGTH/STATUS @(0x08) ; R/W :
→ L'accès en écriture démarre le transfert DMA (SOURCE → DEST).
→ L'accès en lecture retourne l'état du contrôleur DMA (l'état dans l'automate FSM state).
→ L'accès en écriture : idem que SOURCE.
- RESET @(0x0C) ; WONLY :
→ Une écriture sur ce registre interrompt immédiatement le transfert en cours et force l'automate dans l'état IDLE.
→ Une écriture permet aussi d'acquitter les demandes d'interruptions (IRQ).
- IRQ_DISABLE @(0x10) ; R/W :
→ Permet d'activer/désactiver les interruptions.

L'adresse de base du segment associé au contrôleur DMA autant que cible doit être alignée sur une adresse multiple de la taille de l'objet, donc 5 * 4 = 20 octets → 32 octets.

Question B2

L'argument « burst » signifie la taille d'une rafale en nombre de mots.

Question B3

il faut deux automates « MASTER_FSM et TARGET_FSM » pour contrôler le coprocesseur DMA car il peut se comporter comme étant un maitre ou une cible. Le contrôleur DMA joue deux rôles, emmètre et recevoir des commandes, il lui faut donc des états différents pour répondre à cette contrainte.

Question B4

La bascule « r_stop » permet de demander un RESET (logiciel).
La valeur de cette bascule est modifiée dans l'automate cible et testée dans l'automate maitre. En effet, après chaque rafale (Read/Write), l'automate maitre teste la bascule pour arrêter ou pas le transfert en cours.

Question B5

Les conditions de franchissement:
- REQ : le contrôleur DMA émet une requête de lecture/écriture.
- GNT : le bus est alloué au contrôleur DMA.
- LOCK : indique le dernier mot d'une rafale (LOCK = 0).
- LAST : indiquer le dernier mot du transfert.
- ACK : acquittement Pibus, peut prendre 3 valeurs (WAIT, READY ou ERROR).
- RESET : indique une reset logiciel (aller vers l'etat IDLE si activé).

$A' = \overline{REQ}.RESET$
$A = REQ.\overline{RESET}$
$B' = \overline{GNT}$
$B = GNT$
$C = LOCK$
$C' = \overline{LOCK}$
$D = \overline{LOCK}.(ACK = READY)$
$D' = LOCK.(ACK \neq ERROR)$
$D'' = (ACK = ERROR)$
$G = (ACK = WAIT).\overline{RESET}$
$F = (ACK = ERROR).\overline{RESET}$
$E = (ACK = READY).\overline{RESET}$
$F' = (ACK = READY).RESET$

$H = GNT$
$H' = \overline{GNT}$
$I = LOCK$
$I' = \overline{LOCK}$
$J = \overline{LOCK}.(ACK = READY)$
$J' = LOCK.(ACK \neq ERROR)$
$J'' = (ACK = ERROR)$
$M = (ACK = READY).LAST$
$K = (ACK = READY).\overline{LAST}$
$K' = (ACK = READY).RESET$
$L = (ACK = ERROR)$
$N = (ACK = WAIT).\overline{RESET}$

$P = RESET$
$P' = \overline{RESET}$
$Q = RESET$
$Q' = \overline{RESET}$
$O = RESET$
$O' = \overline{RESET}$

C) *Architecture matérielle*

Question C1

La longueur par défaut d'une rafale est de 8 mots.
L'utilisation des grosses rafales permet d'accélérer le transfert.
Augmenter la longueur d'une rafale a pour inconvénient de monopoliser le bus car la demande d'une lecture/écriture se fait sur l'état READ_REQ respectivement WRITE_REQ. Une fois ces états ont été franchis, le processeur reste bloqué jusqu'à le traitement de toute la rafale.

Question C2

- Adresse de base du segment DMA : 0x93000000.
- Le numéro de cible pour le composant BCU : 6.
- Le numéro de maitre pour le composant BCU : 1.
- La ligne d'interruption IRQ contrôlée par le DMA est connectée sur le port d'entrée IRQ_IN[0] du composant ICU.

D) *Application logicielle*

Question D1

L'appel système *fb_sync_write()* n'utilise pas le coprocesseur DMA.
La fonction utilisée pour faire se transfert est *memcpy()* qui est un appel système bloquant.
Cette fonction est implémentée sous forme d'une boucle qui se contente de faire un « Load byte(lb) » suivi d'un « Store byte(wb) ». La main n'est rendu à l'application qu'à la fin du transfert.

Question D2

La durée de construction et d'affichage d'une image **sans** contrôleur DMA :

	Damier 0	Damier 1	Damier 3	Damier 4	Damier 5
Buid time	714252	1551618	2388816	3225985	4063186
Display time	834708	1671925	2509110	3346279	4183480

Total : 4186716.

Question D3

- L'appel systeme *fb_write()* utilise le coprocesseur DMA donc n'est pas bloquant.
Cette fonction utilise la politique de l'attente active pour tester la variable globale *_dma_busy[i]* qui sert à détecter la disponibilité du contrôleur DMA associé au proc[i].
Cette variable est remise à zéro par le handler (ISR) associé à l'interruption DMA (IRQ).
- L'appel système *fb_completed()* sert à détecter la fin du transfert DMA. c'est une fonction bloquante. Le processeur reste gelé jusqu'à réception d'une interruption.

Question D4

La durée de construction et d'affichage d'une image **avec** contrôleur DMA :

	Damier 0	Damier 1	Damier 3	Damier 4	Damier 5
Buid time	714242	1444686	2174723	2904726	3634767
Display time	728549	1458638	2188675	2918678	3648702

Total : 3651968 cycles.

Question D5

Après avoir supprimer l'appel système *fb_completed(),* on observe un dysfonctionnement sur le bord gauche de l'image, l'image n+1 commence à apparaître lors de la phase d'affiche de l'image n. On peut observer plus clairement ce problème en augmentant la taille de l'image (e.x. à 256 pixels).
Le problème provient du fait que l'on touche au contenu du buffer avant que le contrôleur DMA ait fini son transfert.

Question D6

- La fonction *fb_write()* fait une attente passive sur la variable *_dma_busy[i]* pour tester la disponibilité du contrôleur DMA. Une fois le contrôleur est libre, elle le réserve en la remettant à 1.
- Quant à la fonction *fb_completed()*, elle interroge cette variable en bouclant dessus jusqu'à ce qu'elle repasse à zéro.
- La mise à 0 de la variable *_dma_busy[i]* est fait dans l'ISR *_isr_dma()* se trouvant dans le fichier *irq_handler.c.*
- Cette variable est stockée dans le segment *KUNC_DATA*. En effet, elle est déclarée comme variable globale uncachable pour permettre l'interaction avec l'ISR.

E) *Pipe-Line logiciel*

Question E1

Pour passer de l'étape (n) à l'étape (n+1), il faut s'assurer que le contrôleur DMA ait fini de travailler avec le buffer passé en paramètre dans l'étape (n). Donc il faut utiliser la fonction *fb_completed()* pour assurer la synchronisation.

Question E2

La durée de construction et d'affichage d'une image **avec** contrôleur DMA et **pipe-line** :

	Damier 0	Damier 1	Damier 3	Damier 4	Damier 5
Buid time	714278	1430279	2160970	2888417	3618727
Display time	1444524	2164691	2902418	3622454	3645971

Total : 3649300.

Le gain apporté par le parallélisme pipe-line : 3651968 - 3649300 = 2668 cycles.

F) *Traitement des erreurs*

Question F1

Le système interdit que l'adresse du tampon (source/destination) soit dans la zone protégée pour éviter que l'on lit/écrit dans les segments protégés.
Ce type d'erreur doit être détecté avant que la DMA commence à effectuer le transfert parce que le contrôleur DMA ne possède pas de mécanismes pour tester les droits d'accès en mémoire.

Question F2

Le mécanisme qui permet au contrôleur DMA de signaler l'utilisation d'une adresse qui correspond à aucun segment défini est :

- pibus_dma.cpp :
Lorsque le contrôleur DMA reçoit une réponse de type « *bus error* » de la part du contrôleur mémoire, plus précisément, un acquittement *PIBUS_ACK_ERROR*, et que les interruptions sont activées, la fonction *PibusDMA::genMoore()* va activer le signal IRQ sur le port en sortie « *p_irq* ».
Cela a pour effet de générer une interruption qui sera traiter par le processeur en se branchant au gestionnaire d'interruption.
L'automate quant à lui, change d'état en fonction de la transaction en cours d'exécution.

- irq_handler.c :
Le processeur se branche dans la routine de traitement d'interruption *_isr_dma()*, sauvegarde l'état du contrôleur DMA(*DMA_ :SUCCESS, READ_ERROR, WRITE_ERROR, IDLE, RUNNING*) qui correspond aux états de l'automate *MASTER_FSM* dans la variable globale « _dma_status[i] ». Ensuite, il ré-initialise la variable globale « _dma_busy[i] » et acquitte l'interruption en écrivant dans le registre *RESET*.
Les variables globales décrites ci-dessus (_dma_busy, _dma_status) sont un moyen d'interaction entre le logiciel et l'ISR.

- driver.c :
Dans la fonction *_fb_completed()*, et en raison des modifications apportées dans le handler d'interruptions, on sort de l'attente active (boucle) et on teste la variable *_dma_status[i]*. En cas d'erreur, la fonction retourne une valeur différente de zéro(>0).

Question F3

- L'appel système qui signale l'erreur d'une adresse non définie (e.x. 0x0) est : *fb_completed()*. En effet, cette erreur est détecté au moment ou le controlleur Dma effectue le transfert et sera don signalée en réponse à la commande d'écriture.
- L'appel système qui signale l'erreur d'une adresse erronée appartenant à la zone protégée (e.x. 0x80000000) est : fb_write() / fb_read().