Une école de l'IMT

Périphériques, Interruptions & DMA

Guillaume Duc
<guillaume.duc@telecom-paristech.fr>
2015–2016

Plan

Objectifs du cours

- Modèle générique d'interaction avec les périphériques
- Modes de communication
  - Attente active (*Polling*)
  - Interruptions
  - Direct Memory Access (DMA)

Exemple de périphérique

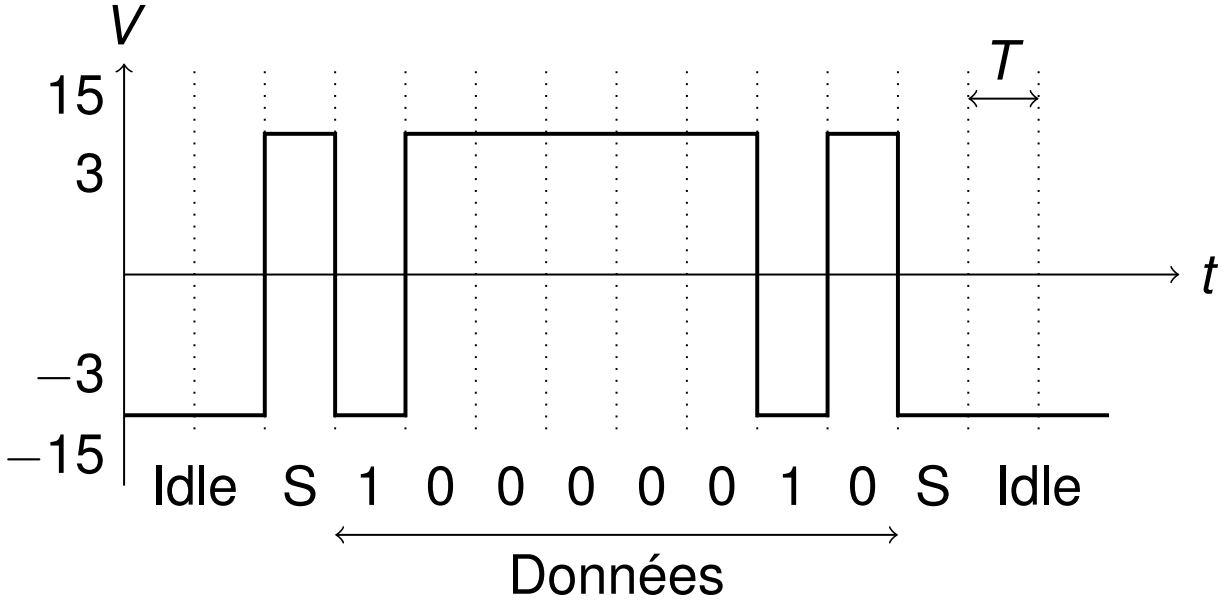
UART

- RS-232 : standard (Electronic Industries Association) pour la communication série asynchrone point à point entre deux équipements (le plus souvent entre un ordinateur et un modem, une souris, une imprimante, etc.)
  - Remplacé de nos jours par l'USB dans les ordinateurs mais encore largement utilisé dans le monde des systèmes embarqués car moins complexe
- UART (*Universal Asynchronous Receiver Transmitter*) : Périphérique gérant la transmission (émission et réception)

Exemple de périphérique

UART — RS-232

- Transmission asynchrone (sans horloge)
- Les données sont transmises en série sur deux fils (un dans chaque sens)
- T = Durée d'un bit, déduit du débit préalablement configuré (exemple : 115 200 bits/s)



Exemple de périphérique

UART — Fonctionnalités nécessaires (non exhaustif)

- Gestion de la communication avec le processeur
- Stockage du caractère à envoyer pendant sa transmission
- Stockage du caractère reçu en attendant son traitement par le processeur
- Gestion de la configuration (débit...)
- Signalisation au processeur d'une réception
- Signalisation au processeur d'une fin de transmission
- Sérialisation / Desérialisation des données
- Génération de l'horloge d'échantillonage et d'émission
- Traitement de la réception (détection du début de la transmission, échantillonage...)
- Traitement de l'émission (ajout des bits de start et de stop...)

Plan

Périphérique générique

- Interface avec le processeur (générique)
  - Données reçues / Données à transmettre
  - État du périphérique
  - Configuration du périphérique
  - Commandes
  - Signalement d'évènements
- Traitements propre au périphérique

Flux d'informations

- Données reçues par le périphérique (périphérique $\rightsquigarrow$ processeur)
- Données à transmettre au périphérique (processeur $\rightsquigarrow$ périphérique)
- État du périphérique (périphérique $\rightsquigarrow$ processeur)
- Configuration du périphérique (périphérique $\leftrightsquigarrow$ processeur)
- Commandes (processeur $\rightsquigarrow$ périphérique)
- Signalement d'évènements (périphérique $\rightsquigarrow$ processeur)

Analogie avec la mémoire

- Lecture par le processeur
  - Données reçues par le périphérique
  - État du périphérique
  - Configuration du périphérique
- Écriture par le processeur
  - Données à transmettre au périphérique
  - Commandes du périphérique
  - Configuration du périphérique
- Notion d'adresse au sein du périphérique pour distinguer les informations à lire/écrire (données/état/configuration/commandes)

Analogie avec la mémoire

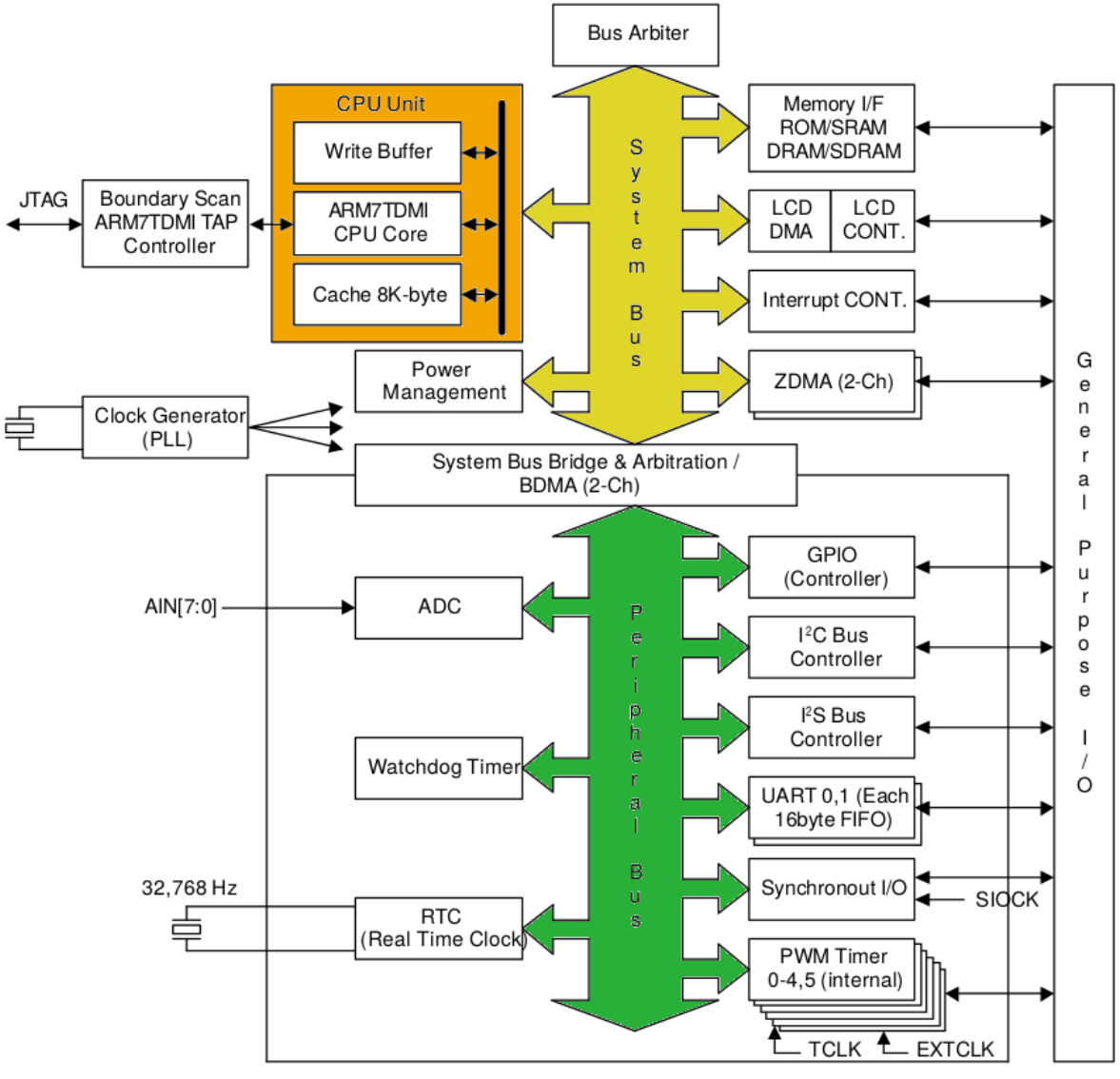
- Du point de vue du processeur, un périphérique peut donc être assimilé à un ensemble de “cases” mémoires, appelées *registres*
- Le contenu de ces registres peuvent correspondre à des données, à l’état, la configuration ou les commandes du périphérique
- Ces registres sont indexés par une adresse
- Pour dialoguer avec le périphérique, le processeur fait des lectures et des écritures dans ces registres

Connexion avec le processeur

- Un périphérique est connecté au processeur grâce à un bus
- Plusieurs cas possibles
  - Un seul bus où sont connectés tous les périphériques et la mémoire
  - Un bus rapide sur lequel est connecté le processeur, la mémoire et un pont (*bridge*) vers un bus plus lent où sont connectés les périphériques
  - Un bus entre le processeur et la mémoire et un bus dédié entre le processeur et les périphériques

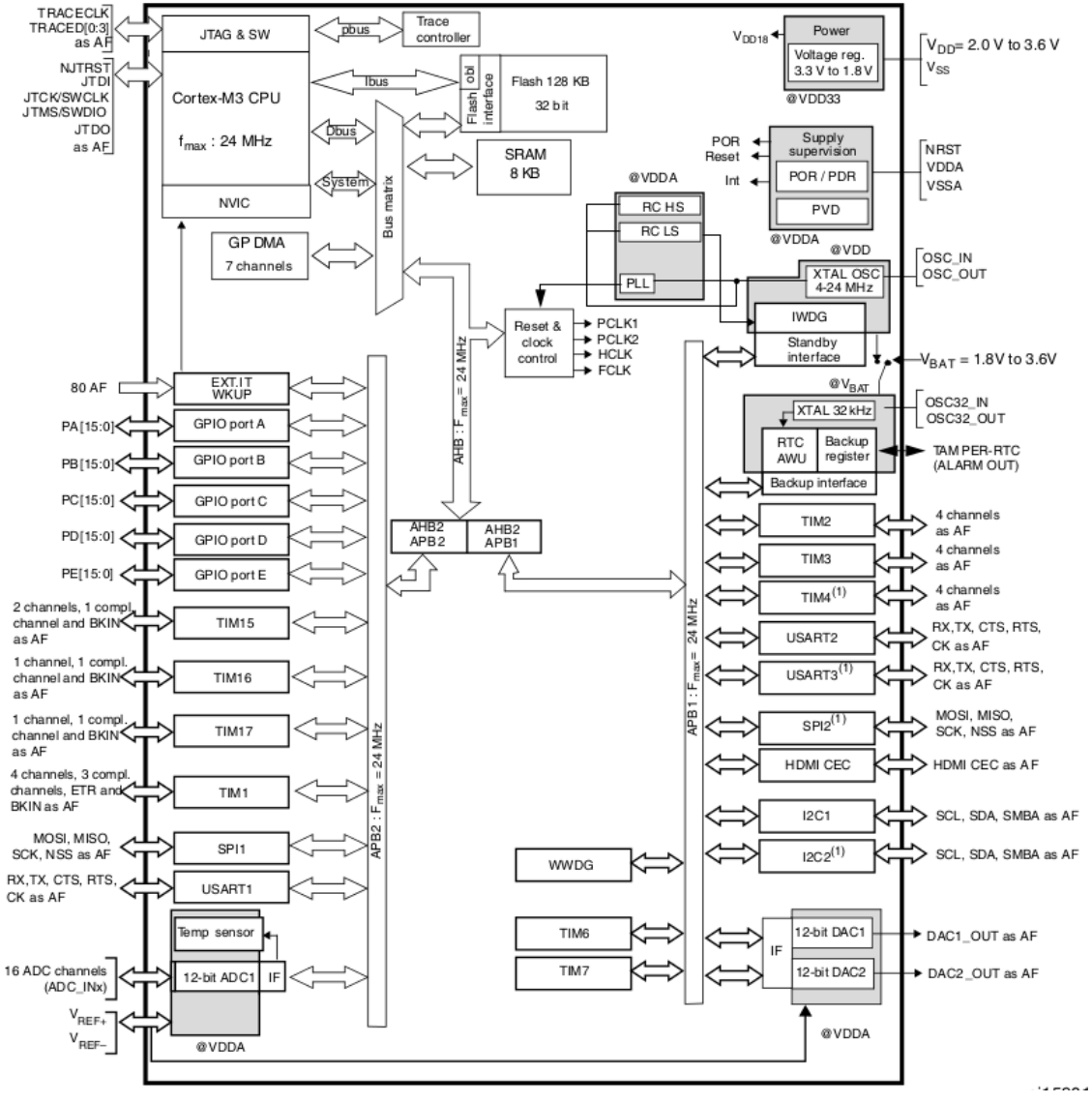
Bus

Samsung S3C44B0X (ARM7TDMI)



Bus

STM32F100 (Cortex M3)



Adressage des périphériques

- *Memory-Mapped I/O (MMIO)*
  - Les adresses des registres du périphérique apparaissent dans l'espace d'adressage du processeur $\rightsquigarrow$ *cartographie mémoire*
  - Les transferts vers et depuis le périphériques (données, commandes, configuration ou statut) se réalisent de façon identique avec les transferts mémoires (ex. instructions *load/store*, modes d'adressage impliquant la mémoire, etc.)
- *Port-Mapped I/O (PMIO)*
  - Deux espaces d'adressage distincts : un pour la mémoire et un pour les périphériques
  - Les transferts vers et depuis les périphériques se réalisent avec des instructions dédiées (ex. instructions `IN` et `OUT` du x86

MMIO vs. PMIO

- *Memory-Mapped I/O*
  - Un seul bus à gérer par le processeur
  - Toutes les instructions qui manipulent des opérandes en mémoire peuvent de fait manipuler des opérandes en provenance de périphériques
  - Contrôle d'accès peut être géré par la MMU (mais granularité d'une page)
- *Port-Mapped I/O*
  - Pas de réduction de l'espace d'adressage qui peut être occupé par de la mémoire (moins un problème avec les architecture 32/64 bits)
  - Contrôle d'accès souvent en tout ou rien (mode superviseur)
  - Instructions spécifiques souvent plus frustres (ex. architecture x86 : `IN` et `OUT` ne peuvent utiliser que le registre `(E)AX` comme source ou destination du transfert)

Plan

Lecture de données depuis un périphérique

Exemple de l'UART du S3C44B0X

UART TX/RX STATUS REGISTER

There are two UART Tx/Rx status registers, UTRSTAT0 and UTRSTAT1, in the UART block.

Register	Address	R/W	Description	Reset Value
UTRSTAT0	0x01D00010	R	UART channel 0 Tx/Rx status register	0x6
UTRSTAT1	0x01D04010	R	UART channel 1 Tx/Rx status register	0x6

UTRSTATn	Bit	Description	Initial State
Transmit shifter empty	[2]	This bit is automatically set to 1 when the transmit shift register has no valid data to transmit and the transmit shift register is empty. 0 = Not empty 1 = Transmit holding & shifter register empty	1
Transmit buffer empty	[1]	This bit is automatically set to 1 when the transmit buffer register does not contain valid data. 0 =The buffer register is not empty 1 = Empty If the UART uses the FIFO, users should check Tx FIFO Count bits and Tx FIFO Full bit in the UFSTAT register instead of this bit.	1
Receive buffer data ready	[0]	This bit is automatically set to 1 whenever the receive buffer register contains valid data, received over the RXDn port. 0 = Completely empty 1 = The buffer register has a received data If the UART uses the FIFO, users should check Rx FIFO Count bits in the UFSTAT register instead of this bit.	0

Lecture de données depuis un périphérique

Exemple de l'UART du S3C44B0X

UART RECEIVE HOLDING (BUFFER) REGISTER & FIFO REGISTER

URXHn has an 8-bit data for received data..

Register	Address	R/W	Description	Reset Value
URXH0	0x01D00024(L) 0x01D00027(B)	R (by byte)	UART channel 0 receive buffer register	–
URXH1	0x01D04024(L) 0x01D04027(B)	R (by byte)	UART channel 1 receive buffer register	–

URXHn	Bit	Description	Initial State
RXDATAn	[7:0]	Receive data for UARTn	–

Lecture de données depuis un périphérique

Exemple de l'UART du S3C44B0X

```
// Version en C
#define UTRSTAT0 (*(volatile uint32_t *) 0x01D00010)
#define URXH0    (*(volatile uint8_t *)  0x01D00024)

while (!(UTRSTAT0 & 0x1));
char recv = URXH0;


// Version en Assembleur ARM
      ldr  r1, =0x01D00010
loop: ldr  r2, [r1]
      and  r2, r2, #1
      cmp  r2, #0
      beq  loop

      ldr  r1, =0x01D00024
      ldrb r2, [r1]
```

Transferts de données

Attente active (*Polling*)

- Déroulement
  - Interroger le registre de statut du périphérique jusqu'à ce qu'une donnée soit disponible ou puisse être envoyée
  - Lire ou écrire la donnée
- Méthode de l'attente active (*Polling*)
- Le processeur doit explicitement aller vérifier si une donnée peut être lue ou écrite et ne peut rien faire d'autre en attendant
- Efficace si les données arrivent rapidement ou s'il n'y a pas d'urgence (vérifications à intervalles longs)
- Alternative : interruptions

Transferts de données

Volatilité des registres des périphériques

- Le contenu des registres des périphériques peut varier sans action explicite du processeur
- Problèmes
  - Caches du processeur : Il ne faut pas que l'espace d'adressage occupé par les périphériques soit cachable (par construction ou par configuration du cache)
  - Optimisations du compilateur : Il faut penser à déclarer les données du périphérique comme *volatiles* pour empêcher les optimisations du compilateur

Plan

Interruptions

- Signal émis par un périphérique indiquant au processeur la survenue d'un évènement
- Asynchrone vis-à-vis du flot d'exécution (peut survenir n'importe quand, non liée à une instruction particulière)
- Le flot d'exécution est interrompu et le contrôle donné à un gestionnaire d'interruption (*Interrupt Handler* ou *Interrupt Service Routine*)
- Une fois le traitement de l'interruption effectué, le flot d'exécution initial reprend là où il s'est interrompu
- Mécanisme pour identifier l'origine de l'interruption (très nombreuses sources dans les systèmes modernes)

Interruptions précises

- Les interruptions en provenance de périphériques doivent être des interruptions *précises*
  - Le compteur de programme est sauvegardé
  - Toutes les instructions précédant le PC sont complètement achevées
  - Aucune instruction suivant le PC n'a été exécutée
- Simplifie la vie du programmeur car il suffit, pour revenir de l'interruption, de reprendre l'exécution à la valeur de PC sauvegardée
- Potentiellement compliqué à réaliser matériellement à cause du pipeline

Déroulement d'une interruption

- Une interruption non désactivée survient
- Le processeur sauvegarde le PC (et quelques autres registres), soit dans un registre particulier, soit dans la pile
- Le cas échéant, le processeur change de mode (en général, passage en mode superviseur)
- Le PC est chargé avec l'adresse du gestionnaire d'interruption
  - Vecteur d'interruption : Adresse de base (fixe ou configurable) + $n \times$ numéro de l'interruption
  - Adresse fixe, le numéro de l'interruption étant chargé dans un registre pour pouvoir être connu

Déroulement (suite)

- Le gestionnaire d'interruption
  - Sauvegarde éventuellement d'autres registres dans la pile pour pouvoir travailler
  - Effectue le travail approprié (ex. transfert les données reçues par le périphériques vers la mémoire)
  - Restaure les registres sauvegardés pour que le programme initial puisse reprendre sans effets de bord
- Retour d'interruption (en général une instruction dédiée), le processeur restaure le PC et le mode d'exécution précédent et le flot d'exécution initial reprend

Premier problème

- Le temps global de traitement de l'interruption ne doit pas être trop important ainsi que la fréquence de survenue des interruptions
- Sinon, le flot d'exécution principal n'a plus le temps de s'exécuter et est constamment interrompu (*Interrupt storm*)
- On risque également de perdre des données reçues par les périphériques

Gestionnaires d'interruption modernes

- Réduction du temps de traitement d'une interruption
  - Le traitement d'une interruption est souvent découpé en deux parties afin de limiter le temps durant lequel le flot d'exécution est interrompu
  - *First-Level Interrupt Handler (FLIH)* : fait le minimum de choses et programme l'exécution du SLIH
  - *Second-Level Interrupt Handler (SLIH)* : effectue le reste du traitement, plus tard, lorsque l'ordonnanceur de tâches du système d'exploitation le décide
- Réduction de la fréquence des interruptions
  - Certains systèmes d'exploitation désactivent temporairement les interruptions (et font du *polling* régulier) pour les périphériques qui en génèrent trop
  - Certains périphériques disposent d'un mécanisme de limitation de la fréquence des interruptions

Réduction du risque de perte de données

- Utilisation de files d'attente (FIFO) dans les périphériques à la place d'un seul registre de réception
- Permet ainsi de stocker plusieurs données reçues en attendant que le processeur vienne les récupérer
- Interruptions associées à la FIFO (en général : FIFO à moitié pleine, FIFO pleine, avec éventuellement des priorités différentes)
- Idem pour les données à transmettre au périphérique (permet de transmettre plusieurs données d'un coup sans avoir à attendre qu'elles soient traitées)

Autre problème

- Même avec les interruptions, le transfert des données depuis le périphérique vers la mémoire (ou vice-versa) reste à la charge du processeur qui ne peut rien faire pendant ce temps
- Solution : le *Direct Memory Access* (DMA)

Plan

Présentation

- Objectif : Décharger le processeur des transferts de données
- Le contrôleur de DMA est un module matériel capable de réaliser des transferts entre les périphériques et la mémoire
- Une fois le contrôleur configuré, le processeur n'intervient plus dans le transfert
- La fin du transfert est généralement signalée par une interruption émise par le contrôleur de DMA
- Avantages
  - Le processeur peut faire autre chose
  - Le transfert des données peut être plus rapide

Configuration minimale

- Adresse de source du transfert
- Adresse de destination du transfert
- Nombre de données à transferer
- Taille des données (octets, mots...)

Déclenchement du transfert

- Déclenchement par logiciel (notamment dans le cas de transferts mémoire vers mémoire)
- Déclenchement sur événement d'un périphérique (exemple : données reçues par la carte réseau)

Contrôleur de DMA

- Plusieurs canaux (*channels*) pour effectuer plusieurs transferts en même temps (en provenance de plusieurs périphériques par exemple)
- Registres de configuration par canaux
- Transferts
  - Périphérique vers mémoire
  - Mémoire vers périphérique
  - Périphérique vers périphérique
  - Mémoire vers mémoire
    - Possibilité de remplir une zone mémoire avec un motif

Modes d'opération

Arbitrage DMA/Processeur

- Rafale (*Burst*) : Le bloc de données est transféré en une seule fois. Une fois que le contrôleur de DMA a accès au bus, il le conserve pendant tout le transfert. Le processeur ne peut pas faire d'accès mémoire pendant ce temps
- Vol de cycle (*Cycle Stealing*) : Le processeur et le contrôleur de DMA se partagent alternativement le bus (un cycle pour le processeur, un pour le contrôleur de DMA)
- Transparent : Le contrôleur de DMA n'a accès au bus que lorsque le processeur n'en a pas besoin

Cohérence des caches

- Problème de cohérence des caches pour la zone mémoire destination du transfert DMA (ou même pour la zone source en cas de politique *write-back* du cache)
- En fonction des architectures
  - Soit le cache est invalidé matériellement
  - Soit il appartient au système d'exploitation d'invalider le cache pour les zones concernées par le transfert

DMA et mémoire virtuelle

- Utilisation des adresses physiques
  - Franchissement d'une frontière de page possible uniquement si données consécutives en mémoire physique
  - Le système d'exploitation ne doit pas toucher aux pages physiques concernées pendant le transfert
  - Plus simple
- Utilisation des adresses virtuelles
  - Nécessite des TLB dans le contrôleur de DMA
  - Peut ainsi gérer des transferts de données consécutives dans l'espace d'adressage virtuel (plusieurs pages)
  - Plus souple mais plus compliqué

Scatter/Gather

- Blocs source/destination non forcément consécutifs (exemple dans le cas d'un DMA utilisant des adresses physiques)
- Contrôleurs récents : possibilité de copier depuis/vers plusieurs blocs non consécutifs
- Au lieu d'une adresse de source/destination + taille, description de plusieurs blocs (adresse + taille) chaînés

Plan

Sources / Références

Sources

- Les images du cours sont issues des documents techniques suivants
  - Samsung S3C44B0X RISC Microprocessor Datasheet
  - ST STM32F100 Datasheet