DROITS DE PROPRIÉTÉ INDUSTRIELLE	PROPRIETARY RIGHTS
Les informations contenues dans ce document sont la propriété de Safran Electronics & Defense et diffusées à titre confidentiel dans un but spécifique. Le destinataire assure la garde et la surveillance de ce document et convient qu'il ne sera ni copié ni reproduit en tout ou partie et que son contenu ne sera révélé en aucune manière à aucune personne, excepté pour répondre au but pour lequel il a été transmis. Cette recommandation est applicable à toutes les pages de ce document.	This document contains information of a proprietary nature to Safran Electronics & Defense company and is submitted in confidence for a specific purpose. The recipient assumes custody and control and agrees that this document will not be copied or reproduced in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered. This legend is applicable to all the pages of this document.

Page laissée intentionnellement blanche.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

_____

Établi par :	Safran Electronics & Defense	VENRIES Thomas	Stagiaire		
Approuvé par :	Safran Electronics & Defense	MAITROT Nicolas	-		

_____

		q
01	10/03/2017	Création du document.
02	19/04/2017	Documentation Technique DRAFT 1
03	09/06/2017	Documentation Technique DRAFT 2/3

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Page

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

_____

Page

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

_____

Q	ontrol **Q**rocess nit	nité entrale raitement
	ardware ssisted irtualization	irtualisation ssistée par atériel
	ardware irtualization	irtualisation atérielle
	ard are	atériel
	perating ystem	ystème d’ xploitation
	uick lator	mulateur apide

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

_____

Dans le cadre de mon stage au sein de SAFRAN Electronics & Defence, ce document technique est rédigé pas à pas, présentant parallèlement l'état d'avancement du stage.

L’objectif principal du stage est l’ajout de l’émulation d’un périphérique propriétaire dans QEMU pour une plateforme cible pré-implémentée. Une application est développée mettant en pratique ce périphérique validant son implémentation fonctionnelle.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Avant de rentrer dans le vif du sujet, il est important de correctement situer Qemu dans un contexte d'émulation, de virtualisation et de simulation. Bien qu'utilisant ces trois techniques de manière transparente pour l'utilisateur lambda, il est parfois difficile de les différencier. Il sera souvent fait mention des termes « invité » et « hôte » correspondant respectivement à la machine virtuelle et la machine physique.

Ce document présente le choix technique qu'est l'émulateur Qemu dans le cadre du stage. Il exposera un cas d'étude de l'émulation actuelle de la Raspberry Pi 2 afin d'illustrer les composantes vitales d'un module « machine » dans Qemu, l'utilisation de son API et son intégration dans le processus de compilation. Cette étude accompagne le développeur débutant dans la compréhension générale du cœur de Qemu.

La procédure d'installation des outils suivants ayant servi à réaliser cette étude sont expliquées pas à pas :

- – Crosstool-ng, a cross toolchain generator
- – bare-metal ARM EABI toolchain (+ newlib)
- – Quick Emulator

Nous assumons dans ce document que :

- Vous avez une base d'installation et une configuration f opérationnelle. Une rubrique Wiki à l'URL est dédiée à une configuration proxy pour les connexions sortantes au sein de l'entreprise SAFRAN ELECTRONICS & DEFENSE.
- Vous avez une expérience Linux et familiarité avec les commandes Shell.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

[A1] Titre
Réf. : SK-XXXXXXXXXX-XX

[R1] VMWare, "*Understanding Full Virtualization, Para virtualization, and Hardware Assist"*. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

[R2] M. Monton, A. Portero, M. Moreno, B. Martinez, et J. Carrabina, "Mixed SW/SystemC SoC emulation Framework", in *Proceedings of IEEE International Symposium on Industrial Electronics*, June 2007, pp. 2338-2341

[R3] T.-C. Yeh, G.-F. Tseng, and M.-C. Chiang, "A fast cycle-accurate instruction set simulator based on QEMU and SystemC for SoC development," in *Proceedings of the 15th IEEE Mediterranean Electro technical Conference*, Apr. 2010, pp. 1033–1038.

[R4] T.-C. Yeh, Z.-Y. Lin, and M.-C. Chiang, "Enabling TLM-2.0 Interface on QEMU and SystemC-based Virtual Platform" in *Proceedings of the 15th IEEE International Conference on IC Design & Technology*, 2011, pp. 1–4

[R5] D. Quaglia, F. Fummi, M. Macrina and S. Saggin, *"Timing aspects in QEMU/SystemC synchronization"*

[R6] J. Fitzgerald, P. G. Larsen, M. Verhoef and K. Pierce, *"Collaborative Design For Embedded Systems – Co Modelling and Co-simulation"*, chapter 2 *"Co-modelling and co-simulation in embedded systems design"*

[R7] Open System C Initiative (OSCI). [Online]. Available: http://www.systemc.org/

[R8] *IEEE Standard System C Language Reference Manual*, 1666-2011.pdf, Design Automation Standards Committee, 2011. [Online]. Available : http://standards.ieee.org/findstds/standard/1666-2011.html

[R9] J. Bismarck, F. Morales, "Evaluating Gem5 and QEMU virtual platforms for ARM multicore architectures", Stockholm, Sweden 2016

[R10] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basy, J. Hestness, D. R. Hower, T. Krisnha, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill et A. Wood, "The gem5 simulator", http://gem5.org/

[R11] QEMU, "*QEMU Monitor – 3.6"*. [Online]. Available: http://download.qemu.org/qemu-doc.html

[R12] Documentation Raspberry Pi Online, "CONFIG.TXT". [Online]. Available: https://www.raspberrypi.org/documentation/configuration/config-txt/

[R13] ARM, "ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition", 2017, pp1532, 1959-1963

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

[R14] Yvan Roch, “qemu : Visite au Cœur de l’émulateur”, chapter 4 “La gestion du temps sous qemu”, GLMF-147, 2011

[R15] Yvan Roch, "qemu : Comment émuler une nouvelle machine ? Cas de l’APF 27", GLMF-148,2011

[R16] The FAQ about BogoMIPS. [Online]. Available: http://tldp.org/HOWTO/BogoMips/bogo-faq.html

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Q

q q

Un hyperviseur est une plateforme de virtualisation qui permet de créer et gérer plusieurs machines virtuelles travaillant en même temps sur une même machine physique hôte.

Il existe deux types d'hyperviseurs présentés en Figure 3.1:

- Hyperviseur de Type-1 ou natif s'exécutant directement sur le matériel de la machine physique.
- Hyperviseur de Type-2 s'exécutant sur un OS hôte comme un programme le ferait.

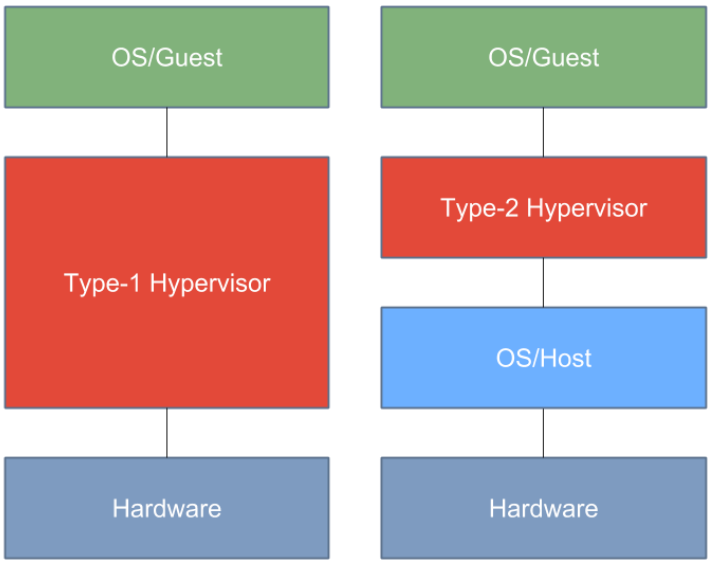


FIGURE 3.1 - HYPERVISORS

Les hyperviseurs utilisent une ou plusieurs techniques de virtualisation [R1] permettant d'utiliser les ressources de la machine physique et de ses périphériques et ce de manière plus ou moins transparente pour la machine virtuelle. Parmi ces techniques de virtualisation, il y en a trois très souvent utilisées :

- Virtualisation Complète (ou en Anglais « Full Virtualization »)
- Virtualisation Assistée par Matériel (ou en Anglais « Hardware-Assisted Virtualization)
- Virtualisation Assistée par OS ou paravirtualisation

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

q est une virtualisation dans laquelle un OS invité non modifié n'est pas conscient d'être exécuté dans un environnement virtuel. Il effectuera ses traitements et commandes à ce qui pensera être le matériel réel. L'hyperviseur virtualise le matériel en effectuant une traduction binaire du code Kernel de l'OS invité et remplace les séquences d'instructions non-virtualisables en nouvelles séquences d'instructions qui aura le même résultat prévu sur le matériel virtualisé. Cette technique permet une très bonne isolation et sécurité des machines virtuelles mais n'offre que de faibles performances.

q est un type de virtualisation complète qui tire parti des fonctionnalités matérielles de certains processeurs comme Intel VT-x et AMD-v ayant pour but d'aider à la virtualisation matérielle offrant des performances accrues parfois proches du natif.

q est une virtualisation dans laquelle l'OS invité est cette fois ci modifié car conscient d'être exécuté dans un environnement virtuel. En conséquence, l'OS invité possède des drivers spéciaux qui au lieu d'envoyer naturellement des commandes primitives et répétitives au matériel, ils envoient des commandes bien plus simples à l'OS hôte qui se charge de les retraduire ensuite au matériel. Cela permet de meilleures performances en comparaison à la virtualisation complète. Des noyaux Linux modernes par défaut embarquent certains drivers spéciaux et passent automatiquement en mode paravirtualisation au démarrage quand ils détectent qu'ils s'exécutent en environnement virtuel.

Il existe aussi deux autres types de virtualisation comme la virtualisation partielle (virtualisation d'une partie du matériel à des fins applicatives) et la virtualisation hybride (paravirtualisation + virtualisation complète) sur lesquels on se passera d'explication car elles sortent du périmètre du stage.

QEMU est à l'origine un émulateur autonome, performant et mature qui dans son milieu, est l'un des plus flexibles et portables en émulant la plus part des ISA commerciaux (ARM, x86, MIPS, PPC, Sparc, Micro Blaze et ETRAX CRIS) sur différentes plateformes hôtes (ARM, x86, MIPS, PPC, Sparc et Alpha). Il peut être exécuté sur un large éventail d'OS (GNU/Linux, MacOS, Windows et certains systèmes UNIX comme BSD et Solaris).

QEMU est un émulateur mais aussi un hyperviseur de Type-2. Il utilise donc les techniques d'émulation et de virtualisation qui sont à différencier :

- L'émulation est une technique permettant d'exécuter un programme ou OS pour une architecture et matériel cible sur une machine physique dotée d'une architecture différente et ne disposant pas de ce matériel. Cette technique est principalement utilisée pour des tests fonctionnels et aide au debug d'applications et d'OS.
- La virtualisation est une technique utilisant uniquement les ressources existantes de la machine physique en offrant une totale abstraction et isolation de celles-ci. Elle permet de créer une VM dans laquelle s'exécute un OS devant être de même architecture que la machine physique. A contrario de l'émulation, les VM sont destinées à la production.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

q L

Dans QEMU l'unique cas où les deux architectures cible et hôte sont les mêmes, QEMU se décharge de son code d'émulation processeur et a recours à la virtualisation à travers l'utilisation d'un accélérateur comme KVM pour atteindre de bien meilleures performances qu'une émulation logicielle complète extrêmement lente. En combinaison avec un accélérateur ou un intégrateur comme XEN, Il peut néanmoins toujours utiliser l'émulation de périphériques (voir Figure 3.2).

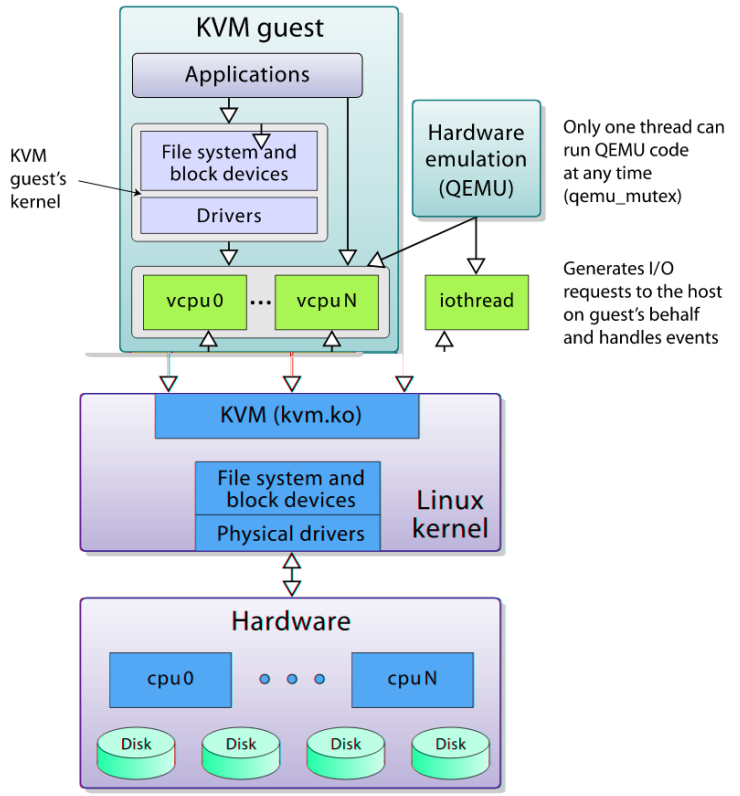


FIGURE 3.2 - FONCTIONNEMENT GLOBAL QEMU/KVM

QEMU seul est très portable mais dans le cas d'utilisation d'un accélérateur comme KVM des contraintes s'imposent. En effet KVM est un module du Kernel (littéralement "Noyau") Linux porté sur certains systèmes UNIX supportant diverses architectures (ARM, x86, PPC et S/390). Il permet au noyau d'avoir la propriété d'hyperviseur offrant toute une infrastructure et interface de virtualisation. KVM utilise comme techniques de virtualisation, la virtualisation assistée par matériel et la paravirtualisation pour les ressources autres que le processeur. Par recours à la virtualisation assistée par matériel, KVM requiert des processeurs avec support de virtualisation. De plus, l'emploi de KVM dépossède QEMU de sa capacité à exécuter un processeur virtuel à différents pas de fréquences (via l'option b , voir la sous-section Annexe 1).

QEMU utilise donc l'API KVM à travers une liaison à l'interface cd liée à au module Kernel (voir Figure 3.2). Historiquement, KVM n'a pas toujours été indépendant comme il l'est actuellement. Il était initialement une branche de développement du projet QEMU appelé d

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

dont le développement a été suspendu car toutes les fonctionnalités de support à KVM ont été ajoutées au projet QEMU original et KVM est maintenant inclus dans le Kernel Linux.

QEMU est souvent accompagné du terme Emulation de Systèmes Complets pour sa spécificité à émuler une collection de matériels existants à usage fonctionnel (et non uniquement du matériel de développement), une architecture à multiprocesseur symétrique et une MMU logicielle (ou SoftMMU) qu'on abordera plus bas (6.2.1).

Dans le contexte de simulation, QEMU est un type de Simulateur de Système Complet (ou Full-System Simulator, FS-S).

QEMU fait partie d'une famille de simulateurs qui sont des Simulateurs d'Architecture (ou Architectural Simulator, A-S). Un A-S est une partie d'un programme ou d'un environnement de simulation qui modélise le composant d'un ordinateur ou son système entier incluant par exemple processeur, système mémoire et périphériques I/O. Un FS-S est donc un type d'A-S qui modélise un système complet.

Un Simulateur à Précision par Cycle (ou Cycle Accurate Simulator, CA-S) est un second type d'A-S qui est une partie d'un programme ou d'un environnement de simulation simulant une microarchitecture avec une précision par cycle. A contrario des FS-S, les CA-S simule un modèle comme le microprocesseur d'un système et non le système entier. Les CA-S sont utilisés quand la précision du temps est primordiale et souvent dans le design de nouveaux microprocesseurs. Ils peuvent être testés, précisément « benchmarké » et dont le design est aisément modelable. Un assemblage de ces modèles peut former un système entier mais sont généralement lents et donc utilisés en combinaison (ou co-simulation [R6]) avec d'autres types de simulateurs principalement pour des raisons de performance.

Les FS-S offrent de nombreux des avantages tels que :

- Exploration d'architecture (opportunité d'ajouter du matériel non-existant au système)
- Vérification fonctionnelle : Testing et debugging
- Plateforme virtuelle de développement logiciel
- Partition HW/SW et optimisation

QEMU a en plus des avantages cités ci-dessus la particularité prépondérante d'être rapide et flexible dans son émulation multi-architectures et portable sur de nombreux OS.

On donnera l'exemple de l'émulateur Bochs, bien qu'aussi portable que son voisin QEMU est loin de posséder ses performances et est uniquement concentré sur l'architecture cible x86. Il a la

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

particularité d'être bâti avec de puissantes built-ins de debug qui font de Bochs un incontournable dans le développement d'OS x86.

- q

Le simulateur Gem5 [R10] est un simulateur d'architecture open-source de haut niveau supporté et utilisé par de nombreuses universités, entreprises et communautés. Gem5 fut développé en tant que Framework de modélisation d'architecture principalement pour la validation de concept et l'étude préliminaire d'exploration. Ce Framework haut niveau intègre un interfaçage Python assurant l'initialisation, la configuration et le contrôle de la simulation, utilise deux langages dédiés C++ (un pour le jeu d'instructions et le second pour la cohérence de cache) et s'applique à un design HW orienté objet alliant des classes Python et C++ (respectivement pour les paramètres et le comportement) et des interfaces standards en Ruby et Garnet.

Comme QEMU, Gem5 supporte de nombreux ISA commerciaux et assure les deux modes d'exécution « System-call Emulation » et « Full-System » présentés en section 1. , mais contrairement à QEMU, il n'assure le boot d'un noyau Linux que sur architectures ARM, ALPHA et X86. En outre, Gem5 s'affirme en framework conçu tout-en-un contrairement à QEMU-SystemC qui est une combinaison de simulateurs avec de nombreux dérivés. Néanmoins QEMU dont les modèles sont entièrement optimisés pour la performance, reste un atout de choix (avec notamment l'emploi de KVM) en termes de vitesse d'exécution dans de telle combinaison de simulateurs face à des frameworks comme Gem5 [R9].

SystemC est une bibliothèque de classes C++ du standard AINSI développée par l'OSCI (Open SystemC Initiative) [R7] en 1999 et approuvée en tant que standard IEEE en 2011 [R8]. SystemC est l'un des langages de modélisation les plus populaires dans le design ESL (Electronic System-Level) dû à sa capacité de simuler concurrence, évènements et signaux hardware.

TLM (Transaction Level Modeling) est une technique de modélisation haut-niveau qui dans le contexte de la modélisation de système digitaux permet de séparer l'implémentation des modules composants de l'implémentation de la communication inter modules. La communication par bus ou FIFO est modélisée sous forme de canaux de communication que les modules composants peuvent utiliser à travers une interface de classes de SystemC. TLM met l'accent sur les détails des transferts de données (quoi et qui) plutôt que sur le détail de l'implémentation. Cela permet d'interagir avec les différents modèles de communication comme les bus à travers une interface commune.

Un outil de simulation ESL comme SystemC offre de nombreux avantages pour le design HW avec ses différents niveaux d'abstraction (TLM) apportant une grande flexibilité de modélisation, des

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

fonctionnalités de profiling et une précision du temps significatives. Cependant, le temps de démarrage d'un OS GNU/Linux Ubuntu 16.04 sur un tel outil est d'environ trente minutes, ce qui est inacceptable.

Pour conserver cette précision et flexibilité qu'offre SystemC tout en accélérant le démarrage d'un OS, plusieurs techniques furent mises en œuvre séparant le processeur (dont l'exécution de l'OS) de la simulation SystemC et déléguer la charge processeur à un autre outil comme QEMU, ce compromis vitesse et précision pouvant démarrer un OS en moins d'une minute.

Le Framework open-source QEMU-SystemC [R2] fut donc mis en œuvre et propose actuellement des fonctionnalités d'émulation HW/SW pour le développement de SoC (System On Chip). Cette stratégie de simulation combine la vitesse de traduction binaire de QEMU avec cette notion et précision du temps apportées par SystemC (sans quoi QEMU n'aurait aucune notion de temps). La communication entre QEMU et SystemC présentée en Figure 3.3 s'opère par l'intermédiaire de sockets modélisant l'interconnexion des composants via un bus AHB. SystemC fait partie intégrante de l'espace d'adressage de QEMU. Le Framework permet à aux périphériques simulés par SystemC d'être insérés à des adresses spécifiques de QEMU et de communiquer via l'interface virtuelle du bus AHB.

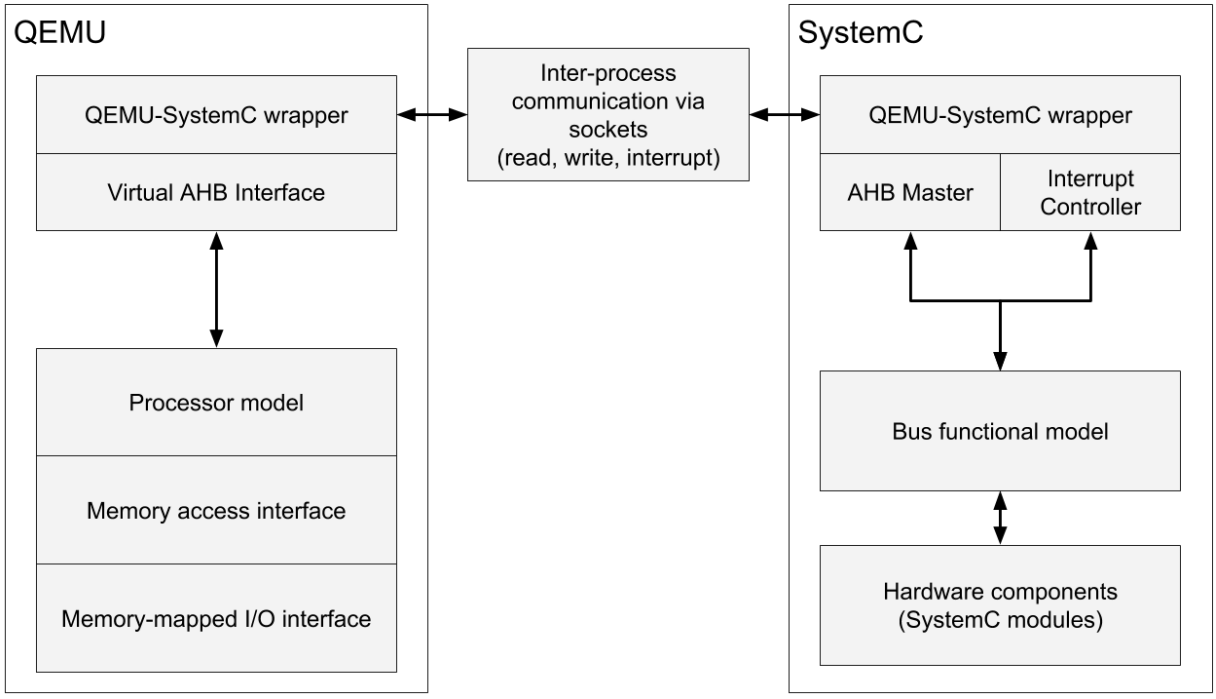


FIGURE 3.3 - QEMU/SYSTEMC CO-SIMULATION FRAMEWORK

Cette communication n'a pas seulement pour but d'échanger des données et des signaux d'interruptions mais aussi pour synchroniser le temps de l'environnement de simulation. Pour cela, un mécanisme algorithmique utilisant différentes approches [R5] est mise en œuvre pour assurer cette synchronisation.

De plus, le Framework peut traquer les opérations entre les composants esclaves SystemC à l'aide les informations transitant sur le bus fonctionnel mais il ne peut pas traquer les transactions du processeur et composants maîtres comme les instructions exécutées, les accès mémoire et ainsi de suite. Néanmoins des solutions existent pour récolter l'ensemble de ces informations [R4].

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Enfin la combinaison QEMU/SystemC peut avoir différentes approches comme une implémentation faible ou forte des composants HW sous SystemC (« faible » signifiant que la majeure partie des composants sont implémentés avec QEMU et le reste en SystemC) et l'utilisation d'interfaces standards.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Tout d'abord, il faut cloner le dépôt de projet d fd:

```
f  b  d      f  c  e  f     R  fd     d     fd f
bc  d     fd
```

- étant un sous module dans notre dépôt d fd, pour télécharger ses sources, on doit entrer la commande suivante :

```
f    a  c  d  c  d      b         f
```

q

- Avant d'installer Crosstool-ng, il se peut que les paquets suivants doivent être installés sur le système : , et .

- La compilation s'effectue en lançant procéduralement les commandes suivantes :

```
bc  b        f
   a
   b  e  f  d    de    GNL D    b
    d
    d
```

On notera que l'option de O SG spécifie le répertoire d'installation dans l'exemple GNL D b .

Dans le répertoire GNL D b on trouve :

- a contenant le script principal
- a contenant le répertoire b f UDQRHNM NLL HS qui inclut les fichiers de configuration, les patchs appliqués aux différents outils et bibliothèques et des scripts Shell.
- d contenant le répertoire b f qui rassemble les archives locales, téléchargées des différents outils et bibliothèques. Ce répertoire n'existe pas par défaut, il est créé pour raison pratique avec une modification de la configuration par défaut de Crosstool-ng en section 4.1.3.

- Pour la complétion Shell, Crosstool-ng offre un script nommé b f b à la racine de ses sources.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Si on dispose de , ajouter la ligne suivante au fichier a b :

Si on dispose de , ajouter la ligne suivante au fichier b :

Les fichiers de configuration portant l'extension se trouvent dans le répertoire GNLD b a b f UDQRHNM NLLHS b e f

On s'intéresse aux fichiers suivants du sous répertoire f a :

- : les chemins d'installation, de build et le répertoire pour les archives locales.
- a c ad d : le choix du Shell dans lequel on effectue la construction des sources, l'option de (pour le nombre de tâche parallèle) par défaut, l'ajout d'options supplémentaires pour le compilateur et linker (EK FR et KCEK FR).
- b ad d : configurations de debug qui accompagne le processus de construction d'une toolchain pas à pas en effectuant des pauses et opérations supplémentaires entre chaque commande réussie et/ou échouée.

Dans le fichier , on modifie :

- KN K S Q KKR CHQ, le répertoire d'accueil des archives téléchargées:

  Ayant choisi GNLD b comme répertoire d'installation pour Crosstool-ng, il arbitraire mais néanmoins pratique de disposer des archives des toolchains à partager dans le même répertoire parent que les configurations et binaire.
- OQDEHW CHQ, le répertoire d'installation des toolchains:

  Les répertoires d'installation des toolchains et leurs outils (sources incluses) sont relativement volumineux (5 à 8 Go) et une fois créés, on les modifie que rarement. Pour cette raison, on les centralise dans le répertoire .

  L'avantage de fixer un répertoire racine d'installation permet à l'utilisateur de configurer une toolchain depuis n'importe quel répertoire. Le principe est illustré section 4.1.4.
- VNQJ CHQ, le répertoire build à créer dans le répertoire d'installation et non dans le répertoire courant:

Dans le fichier a c ad d , on peut modifier:

- O Q KKDK IN R permet d'ajuster l'option a passé à la ligne de commande . Il peut être utile suivant la machine hôte et l'utilisation qu'on en fait, de modifier cette valeur qui est par défaut à 0 ajustant a en fonction du nombre de processeurs sur l'hôte.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Dans le fichier b ad d , on modifie:

- CD TF HMSDQ SHUD active l'option de debug interactif pendant le processus de création d'une toolchain, proposant ainsi à l'utilisateur de réparer ce qui provoque l'erreur d'une commande puis la relancer afin de continuer la construction plutôt que d'arrêter tout le processus qui oblige à reprendre tout depuis le début (par défaut). Pour activer le debug interactif, ajouter la ligne suivante marquée d'un '+':

```
```

Créer un répertoire qui accueillera la configuration d'une nouvelle toolchain:

```
  c        d      b
bc       d      b
```

Avant de générer la configuration d'une toolchain cible, il peut être intéressant d'afficher les informations sur les outils qui la composent avec la commande suivante :

```
b  f                  d a                  SNNK G HM
```

On génère la configuration par défaut pour la toolchain d a :

```
b  f            d a
```

Après cela, plusieurs fichiers et dossiers sont générés :

- a c f : contient les logs du processus de construction de la toolchain
- b e f : lien symbolique vers le dossier de configuration par défaut vu en section 4.1.3.
- b e f fd : la configuration spécifique à la toolchain d a de ses outils, ses bibliothèques et d'autres éléments.
- a c : par défaut placer dans le même répertoire de configuration, sinon dans le répertoire d'installation respectif des toolchains suite à une modification de la configuration par défaut exposé en 4.1.3.
- b e f : le fichier de la configuration générale de la toolchain vu dans d b e f.

On peut modifier la configuration générale du fichier b e f avec la commande suivante :

```
b  f d b e f
```

Enfin, on construit la toolchain avec la commande suivante:

```
b  f a  c
```

On trouvera donc la toolchain installée dans le répertoire d a . Il convient pour une raison pratique d'ajouter le chemin des binaires de la toolchain dans la variable d'environnement O SG.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

La sauvegarde de la configuration peut se faire via les options `db e f` et `dcdeb e f` (cf. 's manpage).

Si vous souhaitez télécharger les sources de `d` depuis le répertoire de projet `d fd` (voir 4.1), mettre à jour le sous-module `f` avec la commande suivante:

```
f a c d c d d
```

Si vous souhaitez cloner le dépôt `d` indépendamment du dépôt de projet :

```
f b d f c e f
```

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

système complet de la forme Q G e. Tentez la force brute en appliquant par exemple une désactivation du mode utilisateur pour une cible du mode utilisateur à compiler entrainera l'erreur suivante DQQNQ T fd d Q G d (plus de détails sur les deux modes d'émulation en section 5.1.1).

- c a dc c : Désactive l'affichage vidéo dans une fenêtre graphique SDL. QEMU utilise deux bibliothèques graphiques SDL et GTK pour l'affichage fenêtré de sa sortie graphique. Désactiver GTK revient à lancer SDL par défaut. Ne rien désactiver, vous laisse le choix par la suite d'exécuter QEMU en mode graphique dans une fenêtre GTK avec l'option suivante c f . Néanmoins sous GNU/Linux, GTK est choisi par défaut. Dans notre cas, nous avons choisi de désactiver SDL car inutile.
- c a d b: On désactivera le support VNC pour la redirection de l'affichage VGA
- d a d cda f: On compile QEMU avec les options de debug afin de pouvoir le lancer avec par la suite.
- d a d bd a b d c d: Active la fonctionnalité basique de tracing qui est mis en pratique dans la section 5.2.2.
- d be f d d : Permet de générer le résultat de traitement du préprocesseur, c'est-à-dire générer des fichiers étendant la gymnastique des macros des sources de QEMU. Cela offrant plus de visibilité sur la finalité du code C.

Dans le cas où on n'a absolument pas besoin de certaines options (réseaux, USB, affichage graphique, accélérateur matériel), on peut désactiver la plus part de ces options avec c a d via le script b e f d. Cela a pour conséquence de rendre l'exécutable (ex : d d plus léger et rapide au lancement sans avoir à modifier le fichier de configuration cde b e f e. Les fichiers sous cde b e f contrôle quel matériel émulé est construit pour chaque cible QEMU en système complet (ex : arm-softmmu) ou utilisateur. Ci-dessous un exemple d'options qui sont généralement peu utilisées :

```
c a d b df f c a d bd c a d b d
c a d f c a d c a d d
c a d dbb c a d a
c a d a a c a d a dc
c a d bf d d d c a d cda f bf
c a d a b c a d ac c a d
c a d b f c a d c a d a
c a d c c a d b b
c a d f d e c a d a e
c a d b c a d a d
c a d d c a d d f
c a d a c a d
```

Avant de compiler QEMU, cela peut être intéressant d'avoir une vue sur le fichier de configuration généré:

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

```
   d   b  e f
```

Enfin, la compilation (cachant l'affichage des commandes exécutées et ne laissant que l'essentiel des logs):

```
      d
```

Lancer l'installation de QEMU :

```
      d
```

A ce stade, l'installation a créé les exécutables d d Q G dans le dossier pré-choisi GNL D b a . Il vous maintenant ajouter ce dossier à la variable d'environnement O SG afin que vous puissiez exécuter QEMU en tapant le nom de son exécutable depuis votre terminal sans avoir à retrouver son chemin. Ajoutez la ligne suivante dans le fichier de configuration de votre Shell, par exemple a b :

```
d       O SG    GNL D      b    a      O SG
```

Dans votre terminal, tapez la commande suivante pour appliquer la ligne ci-dessus:

```
      GNL D  a     b
```

Vous pouvez maintenant exécuter QEMU :

```
   d        d         d
```

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Dans la section précédente, nous avons compilé QEMU uniquement avec deux architectures en mode système complet. Néanmoins il est important de différencier si besoin, les deux modes ou les deux types de binaires que propose QEMU.

QEMU possède deux modes d'émulation:

- Emulation Système Complet (ou Full-system, FS)
- Emulation Mode Utilisateur (ou System-call Emulation, SE)

Le mode FS permet d'émuler un système complet dans une architecture cible et exécuter ce système sur une machine physique pouvant disposer d'une architecture différente. Un système complet peut par exemple émuler une carte type Raspberry Pi avec ses périphériques (ex: RAM, SOC, PIC, RTC et UART) et son processeur ARM dont les divers composants sont actuellement implémentés et disponibles dans QEMU. Le mode FS peut aussi émuler les boot-ROMs. Pour bénéficier du FS, l'option de compilation fd Q G e (où Q G est l'architecture cible) est requise. La compilation produit alors un binaire conventionnellement nommé d d Q G.

Le mode SE permet d'exécuter sur un OS hôte (disposant d'un Kernel Linux uniquement), un binaire d'une architecture différente de l'hôte. Par exemple, QEMU exécutant un binaire ARM sur un OS GNU/Linux x86 ou exécutant un binaire MIPS sur un OS GNU/Linux ARM. Pour bénéficier du SE, l'option de compilation fd Q G d est requise. La compilation produit alors un binaire conventionnellement nommé d Q G.

Pour le reste du document, nous utiliserons QEMU uniquement en mode FS.

q

Vous trouverez ci-dessous une ligne de commande récurrente pour le déploiement d'un applicatif bare-metal sur la machine pour son exécution simple sans le recours au serveur que nous expliquerons en 5.2.1.

```
d   d   L   R
  c f d  d   bd d d   d d
    f    b  d          c
   b      d  d  e  d e
```

- L'option L lance l'émulation de la Raspberry Pi 2. QEMU s'impose grâce à ses nombreux contributeurs avec une liste importante de cibles matérielles dans sa collection. La liste s'obtient avec la commande suivante :

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

```
d       d       L
```

De même, pour une machine cible, on peut lui imposer un processeur en option. On peut lister les processeurs disponibles pour l'architecture ARM, en entrant la commande suivante:

```
d       d       L       b
```

- Les options `c f d d` et `bd d d d d` servent respectivement au log QEMU et d'évènements que nous aborderons dans les sections 5.2.3 et 5.2.2.

- Par défaut QEMU émule une carte VGA peu importe la cible. L'option `f b` désactive la redirection de l'affichage VGA dans une fenêtre graphique. Cela nous évite d'ouvrir une fenêtre graphique au lancement de QEMU d'autant plus que l'on utilisera uniquement l'émulation de l'interface UART de la cible pour l'affichage des . La sortie UART peut être redirigée sur le terminal virtuel de l'hôte qui a servi à lancer QEMU (ou sortie standard, `c`), sur une interface de l'hôte (c.à.d. un autre terminal virtuel) ou sur le réseau (avec par exemple l'option `b c`).

- L'option `d` redirige la sortie UART de la cible sur une interface dite « pseudo terminal » ou . Sous Linux, un pseudo terminal est une paire de périphériques virtuels en mode caractère qui offre une communication bidirectionnelle maitre-esclave où dans ce cas-ci le maitre est le périphérique UART de la cible et l'esclave une interface qui se comporte comme un terminal classique. Pour illustrer cette définition, sous Linux, chaque terminal virtuel ouvert sur une session est lié à une interface que l'on retrouve numérotée dans le répertoire `cd`. L'option ci-dessus va donc créer une nouvelle interface à laquelle il faut se connecter avec des outils d'émulation de terminal comme ou . On choisira au-delà de sa richesse d'options, pour sa capacité à se reconnecter automatiquement à une interface qui a cessé d'exister puis qui réapparait. En effet, avant que le processus QEMU s'arrête, il se détache de l'interface en la supprimant, cela a pour effet sur certains outils comme de devoir l'arrêter et attendre de relancer un nouveau processus QEMU avant de s'y reconnecter en relançant la commande dans un terminal.

  Pour se connecter à l'interface via , on entre la commande suivante dans un terminal:

```
b   C  cd     W            W        cd     d e bd
```

- Précédemment, nous avons connecté la sortie UART de la cible sur un différent de la sortie standard. Celle-ci étant libre, nous allons l'utiliser pour la fonctionnalité de monitoring de QEMU par l'ajout de l'option `c` (pour plus de détail, voir 5.1.3).

- L'option `b` sert à imposer une vitesse au processeur de la cible. Nous aborderons son utilisation en section 5.3.

- L'option `d d e d e` permet de charger le programme dans la mémoire physique de la cible à l'adresse indiqué dans le fichier objet au format ELF.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

QEMU possède une console de monitoring [R11] offrant l'usage d'un jeu de commandes complexes. Elle permet principalement d'interagir avec la VM pendant son exécution (par exemple et b), lui ajouter ou supprimer des périphériques à chaud (voir 1.1), l'inspecter (sans debugger externe) avec e pour récupérer la liste de ses périphériques actifs, l'arbre de ses blocs mémoires et les mappings IO, ses interruptions matérielles ou encore savoir comment s'organise les périphériques sur le bus système.

Pour plus d'information sur une commande moniteur, on tape la commande suivi de d :

```
  d      e   d
```

L'utilisation des commandes moniteur sera mise en pratique dans la section 5.1.3.

Pour lancer rapidement un serveur avec QEMU, voici la commande classique :

```
  d      d      NOSHNMR   R
```

- L'option est une abréviation pour de l'option fca b suivi de R pour stopper le CPU au lancement de la VM. Il se relancera à la connexion d'un client .

Si l'on souhaite que son programme soit chargé depuis QEMU, on utilisera l'option d d :

```
  d      d      NOSHNMR    d  d  e   d e  R
  SNNK G HM fca e   d e
    fd   d   d
  a d
```

Si non avec le client :

```
  d      d      NOSHNMR   R
  SNNK G HM fca e   d e
    fd   d   d
    c           b   fd d   c  a    d
  a d
```

Au sein du dépôt de projet d fd, chaque dossier d'applicatifs bare-metal situés dans le répertoire d contienne un fichier fca b qui sont à utiliser avec de la manière suivante :

```
  SNNK G HM fca    fca b
```

Ces scripts permettent de debugger rapidement les applicatifs compilés de cette façon :

```
    d cda f
```

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Q

La commande ci-dessous introduit l'activation de la fonctionnalité de tracing au lancement de QEMU. Il faut s'assurer que l'on a bien compilé QEMU avec l'option d a d bd a b d c d (voir section 0) :

```
d d NOSHNMR bd d d d d e d f
```

Pour avoir une bonne compréhension du fonctionnement du tracing, un rapide exemple

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

- Il ne reste plus qu'à activer cet évènement. Pour cela, il faut écrire dans un fichier nouvellement créé, le nom de l'évènement :

```
    db    ab        d    b        d d
```

- Puis on lance QEMU avec la commande ci-dessous où d d d d référence le fichier contenant la liste des évènements à activer et e d f le fichier qui contiendra le résultat d'affichage de la fonction de tracing. Ne pas utiliser le paramètre e d , revient à afficher sur la sortie standard (en créant malgré tout un fichier bd dans le répertoire courant) :

```
 d      d      NOSHNMR          c
          bd d d       d d   e  d      f
```

- En mode monitor, on tape la commande suivante afin de vérifier que notre évènement est bien actif (1 : actif, 0 : inactif) :

```
  d     e    bd d d   ab        d    b
ab        d    b       d
```

L'évènement peut manuellement être activé en mode monitor à l'aide de la commande suivante :

```
  d      bd d d   ab        d    b
```

- Enfin le processus QEMU se termine, il nous faudra appliquer un script sur le fichier f pour rendre Pson contenu lisible. Le script en question est situé dans le dossier b à la racine des sources de QEMU:

```
    b         d   bd            bd d d          f
ab        d    b    S  d    b  e
```

En mode monitor, la première commande permet de connaitre l'état de tous les évènements,

```
  d     e    bd d d
```

Lister tous les évènements implémentés :

```
   d      d      NOSHNMR     bd
```

La commande ci-dessous introduit l'utilisation des logs de QEMU. L'option c d f d f permet d'activer seulement certains types de log et C le fichier dans lequel seront écrit les messages. f d d permet d'afficher les logs d'erreur.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

```
d       d       NOSHNMR   c f d   d       C  EHKD
```

Les logs de QEMU offrent une mine d'informations sur l'état des registres CPU pendant et avant le de la VM ou encore avant l'exécution d'un bloc code traduit, l'inspection des blocs binaires traduits affichés en Assembler et l'affichage les interruptions et exceptions interceptées.

Pour lister tous les types de log, exécuter la commande suivante :

```
d       d       NOSHNMR   c
```

Q

Quand Qemu est lancé avec l'option b M, pour chaque instruction du processeur virtuel exécutée, l'horloge virtuelle de la VM nommée (introduit en section 6.3), sera incrémentée de M. Ainsi, pour chaque instruction, avec :

- b : VIRTUAL_CLOCK est incrémentée de 2^0 = 1 nanoseconde.
  - ⇨ Le processeur virtuel a une cadence virtuelle de 1 GHz.
- b : VIRTUAL_CLOCK est incrémentée de 2^1 = 2 nanosecondes.
  - ⇨ Le processeur virtuel a une cadence virtuelle de 500 MHz.
- b : VIRTUAL_CLOCK est incrémentée de 2^3 = 8 nanosecondes.
  - ⇨ Le processeur virtuel a une cadence virtuelle de 125 MHz.

[R17] Le mécanisme d'incrémentation de est illustré en Figure 6.4 et très bien expliqué dans le document d'Yvan Roch, "qemu : Visite au Cœur de l'émulateur", chapter 4 "La gestion du temps sous qemu", GLMF-147, 2011

[R18] Yvan Roch, "qemu : Comment émuler une nouvelle machine ? Cas de l'APF 27", GLMF-148,2011

[R19] The FAQ about BogoMIPS. [Online]. Available: http://tldp.org/HOWTO/BogoMips/bogo-faq.html

dont le Tableau 5.1 en est extrait.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Type de processeur hôte et vitesse en BogoMIPS (provenant de /proc/cpuinfo)	Machine 1 : Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz 4662 BogoMIPS (x4)	Machine 2 : Intel(R) Xeon(TM) CPU 1.70GHz 3361 BogoMIPS
Temps écoulé sur le système hôte sans -icount	21s	39s
Temps écoulé sur le système invité sans -icount	0m20.297s (varie un peu d'un lancement à l'autre)	0m38.306s (varie un peu d'un lancement à l'autre)
BogoMIPS invité sans -icount	304.74 BogoMIPS	172.85 BogoMIPS
Temps écoulé sur le système hôte avec -icount 0	22s	68s
Temps écoulé sur le système invité avec -icount 0	0m7.003s (toujours fixe)	0m7.003s (toujours fixe)
BogoMIPS invité avec -icount 0	999.42 BogoMIPS	999.42 BogoMIPS
Temps écoulé sur le système hôte avec -icount 1	22s	68s
Temps écoulé sur le système invité avec -icount 1	0m14.010s (toujours fixe)	0m14.010s (toujours fixe)
BogoMIPS invité avec -icount 1	499.71 BogoMIPS	499.71 BogoMIPS
Temps écoule sur le système hôte avec -icount 4	24s	74s
Temps écoulé sur le système invité avec -icount 4	1m52.567s	1m52.563s
BogoMIPS invité avec -icount 4	62.25 BogoMIPS	62.25 BogoMIPS

TABLEAU 5.1 - INFLUENCE DE L'OPTION –ICOUNT SUR L'HORLOGE DU SYSTEME INVITE

Le Tableau 5.1 représente une expérimentation de l'option b sur deux machines ayant des vitesses d'exécutions différentes et mesurées en BogoMIPS (million d'instructions par seconde) 0. La mesure s'obtient par lecture du fichier b b e sur tout système Linux.

L'intérêt de b est de simuler de façon très simple le temps qui passe sur la cible émulée en fonction d'une cadence processeur choisie permettant ainsi de cloisonner ce temps à l'environnement d'émulation. De cette façon le temps d'exécution d'un programme est le même peu importe la machine hôte (voir Tableau 5.1).

Sans b, le temps dépendrait entièrement de la machine hôte, impliquant que:

- le temps d'exécution d'un programme sur l'émulation inclut l'exécution de Qemu pour l'émulation elle-même.
- Le temps dépend du matériel de la machine physique hôte.

L'exécution d'outils de benchmarks sur émulation comme Coremark Pro présenté en 5.3.2, nécessite obligatoirement l'emploi de l'option b pour la cohérence des résultats.

On ajoute que l'option b dispose de paramètres pour le « record/replay » une fonctionnalité disposant de fonctions utilisées pour rejouer l'exécution de QEMU dite « déterministe ». L'implémentation permet un debugging déterministe du code exécuté sur le système invité à travers un outil comme (pour plus de détails, lire la documentation interne c b d).

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

q Q

Coremark est un benchmark synthétique qui mesure la performance des CPUs des systèmes embarqués. Il fut développé en 2009 par EEMBC et destiné à devenir un standard d'industrie.

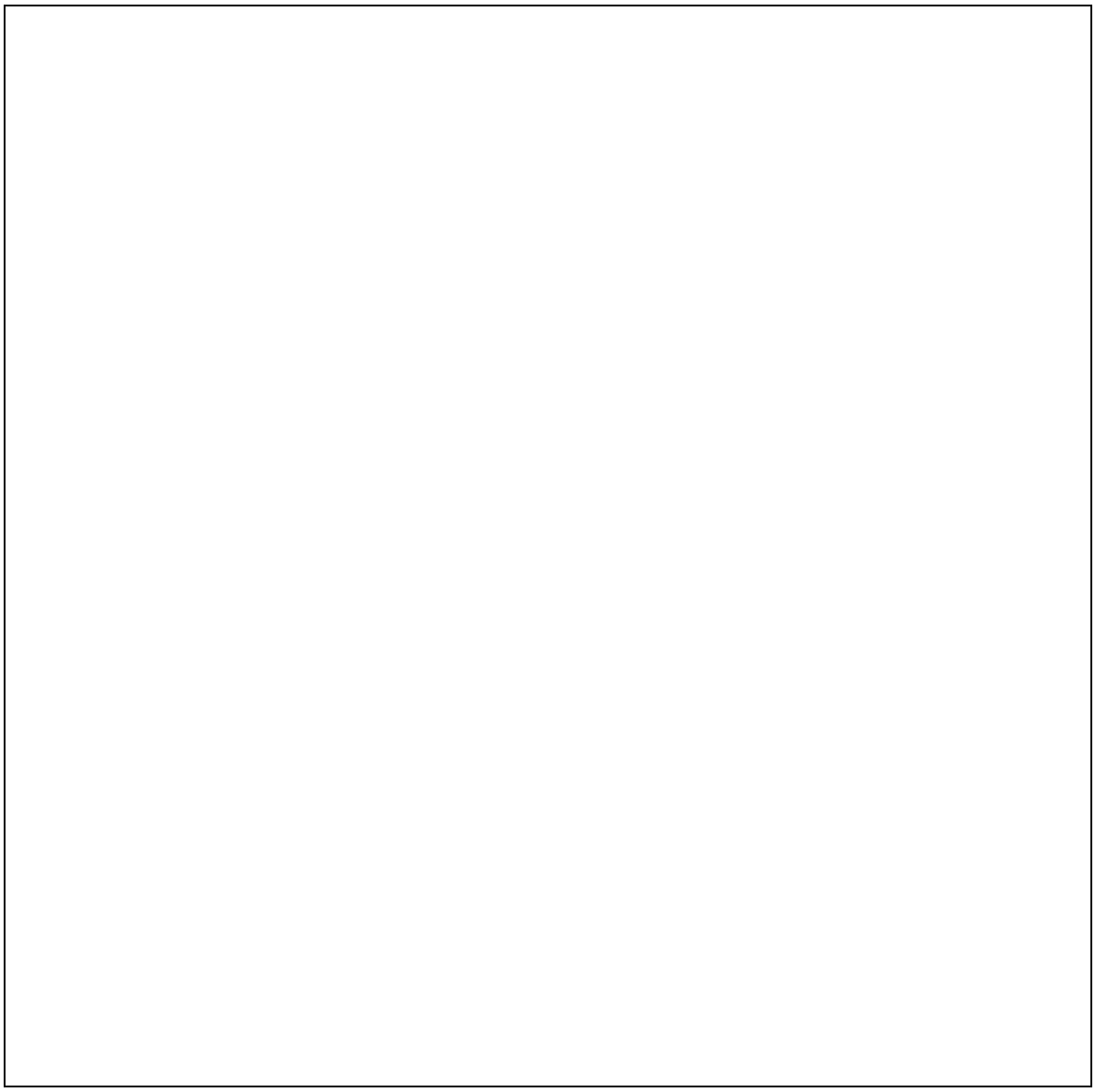
Coremark est écrit en C et contient une implémentation d'une suite d'algorithmes de traitement sur listes (find et sort), manipulations matricielles (opérations courantes), automates finis (déterminant si une entrée contient des nombres valides) et CRCs.

Deux tests de portage sur l'émulation de la Raspberry Pi 2 furent effectués, l'un avec sa version simple et l'autre Pro (embarquant plus d'algorithmes que sa version simple pour ces benchmarks). On retrouve ces tests dans le dépôt projet d fd, dans le répertoire d b d et d b d . Le portage de Coremark Pro est expliqué en section 5.4.1.

Observation des résultats en incrémentant le paramètre de l'option b :

- Un premier essaie de Coremark Pro sur l'émulation de la Raspberry Pi 2 avec l'option b qui pour rappel signifie de disposer d'un processeur cadencé à 250 MHz :

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.



Les résultats l.29 en rouge montrent que le benchmark a pris environ 7.1 secondes.

- Un second essai sur la même cible avec l'option `b` (fréquence : 125 MHz):

- Un troisième essai sur la même cible avec l'option `b` (fréquence : 62.5 MHz):

- Enfin un dernier essai sur la même cible sans l'option `b` donc dépend de la machine hôte :

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

A partir des trois premiers essais, on constate qu'en divisant la cadence du processeur consécutivement par deux (donc en incrémentant b de 1), Coremark fournit des temps multipliés par deux, ce qui est consistant. Le Tableau 5.2 centralise les résultats obtenus :

q -icount	Q	q	q
b	250	7.1	33.4
b	125	14.2	33.4
b	62.5	28.5	33.4
Sans b	~	33.4	33.4

TABLEAU 5.2 - RECAPITULATIF DES RESULTATS DE COREMARK PRO AVEC ET SANS –ICOUNT

Le portage bare-metal de Coremark Pro sur l'émulation de la Raspberry Pi 2 en section Annexe 1 nécessitait une source de temps pour s'exécuter correctement. Le portage a donc introduit en premier lieu une étude de l'architecture de la carte réelle afin de lister les timers et autres compteurs libres utiles puis une analyse de l'existence de leur implémentation dans QEMU.

Le SoC BCM2835 possède au total cinq timers répartis sur deux périphériques :

- Le bloc System Timer : 4 x timers (dont deux disponibles)
- Le bloc ARM SP804 Timer : 1 x timer

Le timer ARM a contrario du System Timer ne possède pas une horloge dérivé directement du quartz mais de celle du cœur GPU qui est sujette à des fluctuations de fréquences dans le cas où le cœur serait en « *reduced power mode* ». La documentation précise qu'il est préférable d'utiliser le System Timer.

Seulement deux timers du ST sont disponibles, les deux autres étant réservés par défaut au GPU. Dans les deux disponibles, l'un est réservé au noyau Linux et l'autre n'est pas alloué donc disponible.

Le ST possède un fonctionnement particulier expliqué en section 7.2.1.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Le processeur ARM possède au total treize timers :

- Les Generic Timers : 4 x timers par cœur (dont 3 disponibles)
- Le Local Timer

Les Timers du processeur possèdent des horloges directement dérivées du quartz sont stables et généralement plus appréciés à l'emploi que ceux du System Timer. Les timers du processeur fonctionnent sur le principe du décompte jusqu'à 0 puis lèvent une interruption. A contrario ceux du ST ont un déclencheur contrôlé selon une comparaison de leur valeur avec les bits de poids faibles de celle d'un compteur libre. Néanmoins le noyau Linux s'appuie sur le ST comme source de temps stable et possédant une résolution suffisamment importante pour son horloge KN J LNMNSNMH et compteur atomique.

De tous les timers précédemment énoncés, aucun de ceux du SoC BCM2835 n'est implémenté dans QEMU. Néanmoins l'émulation des processeurs des Cortex ARMv7-A est correctement implémentée et permet l'utilisation des Generic Timers, Local Timer et compteurs libres inclus dont on tira parti pour Coremark Pro en section Annexe 1.

Q Q

L'essentiel du portage sur l'émulation était de trouver une source de temps pour Coremark comme un compteur libre. Ne disposant pas de l'implémentation des compteurs du SoC BCM2835, il ne restait que l'ARM System Counter (noté SC) de l'implémentation ARMv7-A (voir). Ce SC est hérité des Generic Timers [R13] qui est une extension optionnelle de l'implémentation d'un processeur ARMV7-A.

Pour fonctionner correctement Coremark a besoin de :

- Savoir lire la valeur du compteur
- Connaitre la fréquence du compteur (valeur fixe dans le code) ou savoir la lire

Pour une raison encore inconnue, QEMU fixe la valeur de la fréquence du compteur à 65.2 MHz par défaut.

- La valeur du compteur de System Counter est codée sur 56 bits mais lors de sa lecture, la valeur retournée est un 64-bit complété par des zéros sur les 8 derniers bits de poids forts. La lecture du compteur s'effectue via les instructions suivantes :

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Son équivalent en instruction C/ASM :

- Sa fréquence est lue avec l'instruction suivante :
- Sa fréquence est réécrite avec l'instruction suivante :

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

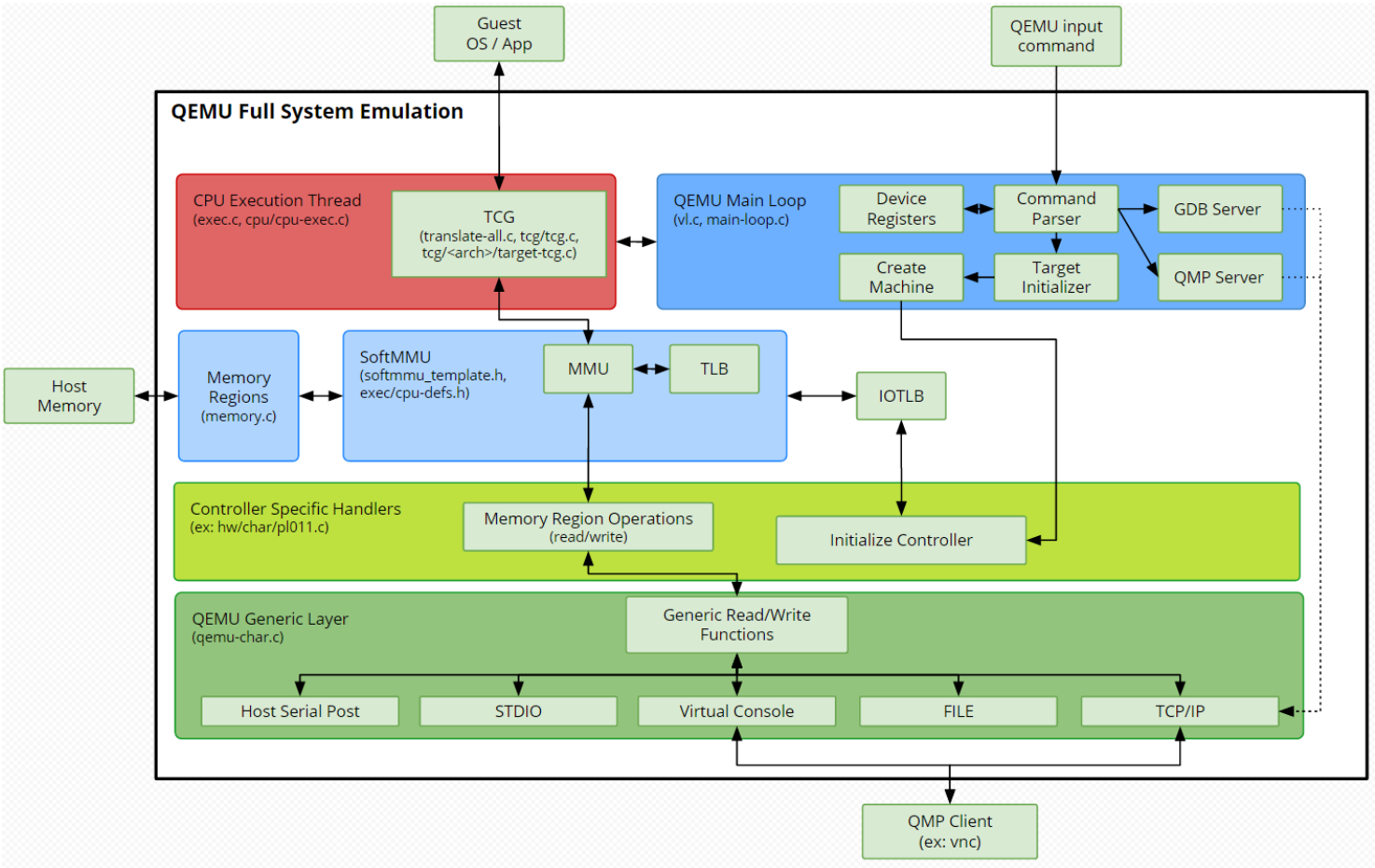


FIGURE 6.1 - DIAGRAMME EN BLOCS D'ARCHITECTURE GLOBALE DE QEMU

A l'exécution de QEMU, il commence avant toute chose par référencer dans une liste doublement chaînée l'ensemble de ses modules périphériques et modules machines. Il analyse ensuite la commande donnée en entrée par l'utilisateur. A la suite de cela, les configurations sont effectuées et le module machine cible est initialisé et créé. Le module machine va à son tour initialiser l'ensemble de ses modules périphériques qui vont être mappé en mémoire.

La boucle main de QEMU utilise une architecture hybride entre une architecture parallèle et « event-driven » architecture pour la gestion des évènements.

La boucle main de QEMU va créer à la suite de l'initialisation du modèle machine, un thread processeur.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Q

Le thread processeur qui émulera un processeur virtuel se chargera avec le TCG (Tiny Code Generator) d'effectuer une traduction des instructions de l'architecture cible en instructions de l'architecture hôte. Ce thread se chargera comme un processeur réel d'exécuter les instructions cibles stockées dans un système de mémoire dédié appelé « régions mémoires » en passant par une MMU logicielle.

La SoftMMU est une implémentation logicielle d'une MMU que QEMU utilise pour accélérer le processus de recherche de « *mapping* » mémoire entre l'adressage virtuelle ou physique du guest et l'adressage virtuel de l'host, de même entre les régions I/O du guest et les fonctions I/O émulées par QEMU.

QEMU utilise donc la SoftMMU comme accélérateur avec comme idée de stocker un « offset » de différence entre l'adresse virtuelle du guest stockée dans la TLB et l'adresse virtuelle de l'host (voir Figure 6.2 - principe d'acceleration de recherche de la SOFTMMU).

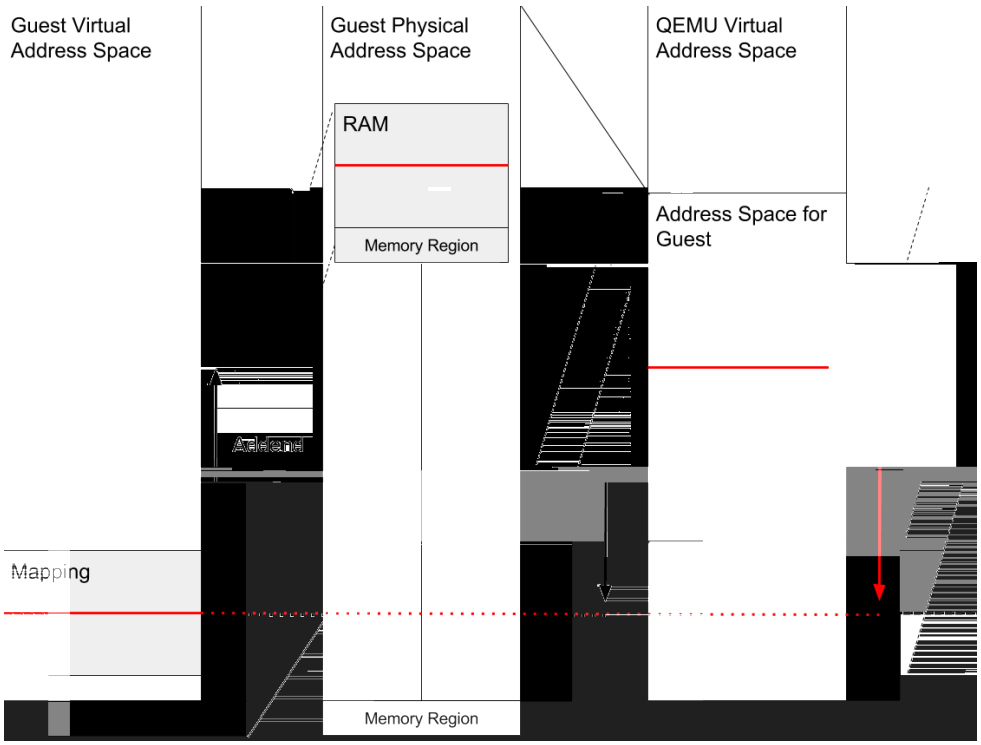


FIGURE 6.2 - PRINCIPE D'ACCELERATION DE RECHERCHE DE LA SOFTMMU

Sans ce modèle de SoftMMU, on serait contraints de traduire en permanence l'adresse virtuelle du *guest* en adresse physique puis l'adresse physique du *guest* en adresse virtuelle du host.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

QEMU offre une API mémoire pour la modélisation et la gestion de la mémoire ainsi que les bus I/O et différents contrôleurs du système émulé. La modélisation s'étend à :

- la RAM (Random Access Memory),
- les Memory-mapped I/O (MMIO),
- les contrôleurs mémoire.

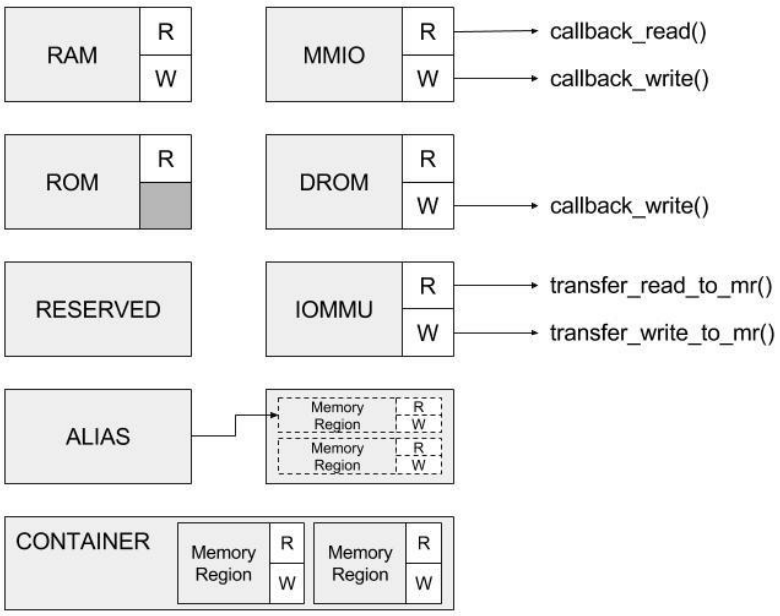


FIGURE 6.3 - LES REGIONS MEMOIRES

La modélisation mémoire offre le support de traçage des changements RAM du système invité.

L'API permet la création de différentes régions mémoires schématisées en Figure 6.3:

- un bloc RAM
- un bloc MMIO
- un bloc I/O Memory Management Unit (IOMMU)
- un conteneur
- une sous-région
- un alias
- un bloc mémoire réservé

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

QEMU dispose de trois différentes horloges :

- Horloge temps réelle nommée PDLT KN J QD KSHLD :

  Utilise la fonction b b fd d de Linux sur l'horloge KN J LNMNSNMH .

  Donne le nombre de nanosecondes écoulées depuis le démarrage de la machine hôte.

- Horloge hôte nommée PDLT KN J GNRS :

  Utilisée pour connaitre le temps du système hôte.

  Donne nombre de microsecondes écoulées depuis l'époque Unix.

- Horloge virtuelle nommée PDLT KN J UHQST K :

  Utilisée comme référence pour la VM.

[R20] Donne le nombre de nanosecondes écoulées depuis le démarrage de la VM qui s'appuie soit soit sur l'horloge hôte, soit sur un mécanisme spécifique d'incrémentation dans le cas d'utilisation de l'option b (décrite et mise en pratique en section 5.3). Ce mécanisme est illustré en Figure 6.4 et très bien expliqué dans le document d'Yvan Roch, "qemu : Visite au Cœur de l'émulateur", chapter 4 "La gestion du temps sous qemu", GLMF-147, 2011

[R21] Yvan Roch, "qemu : Comment émuler une nouvelle machine ? Cas de l'APF 27", GLMF-148,2011

[R22] The FAQ about BogoMIPS. [Online]. Available: http://tldp.org/HOWTO/BogoMips/bogo-faq.html

- Horloge virtuelle temps réelle nommée PDLT KN J UHQST K QS :

  Sans l'option b cette horloge est la même que l'horloge virtuelle.

  Avec l'option b , cette horloge compte les nanosecondes écoulées pendant le fonctionnement de la machine virtuelle en se basant sur PDLT KN J QD KSHLD. Elle est utilisée pour augmenter le compteur de l'horloge virtuelle quand les CPUs sont endormis et donc n'exécutant aucune instruction.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

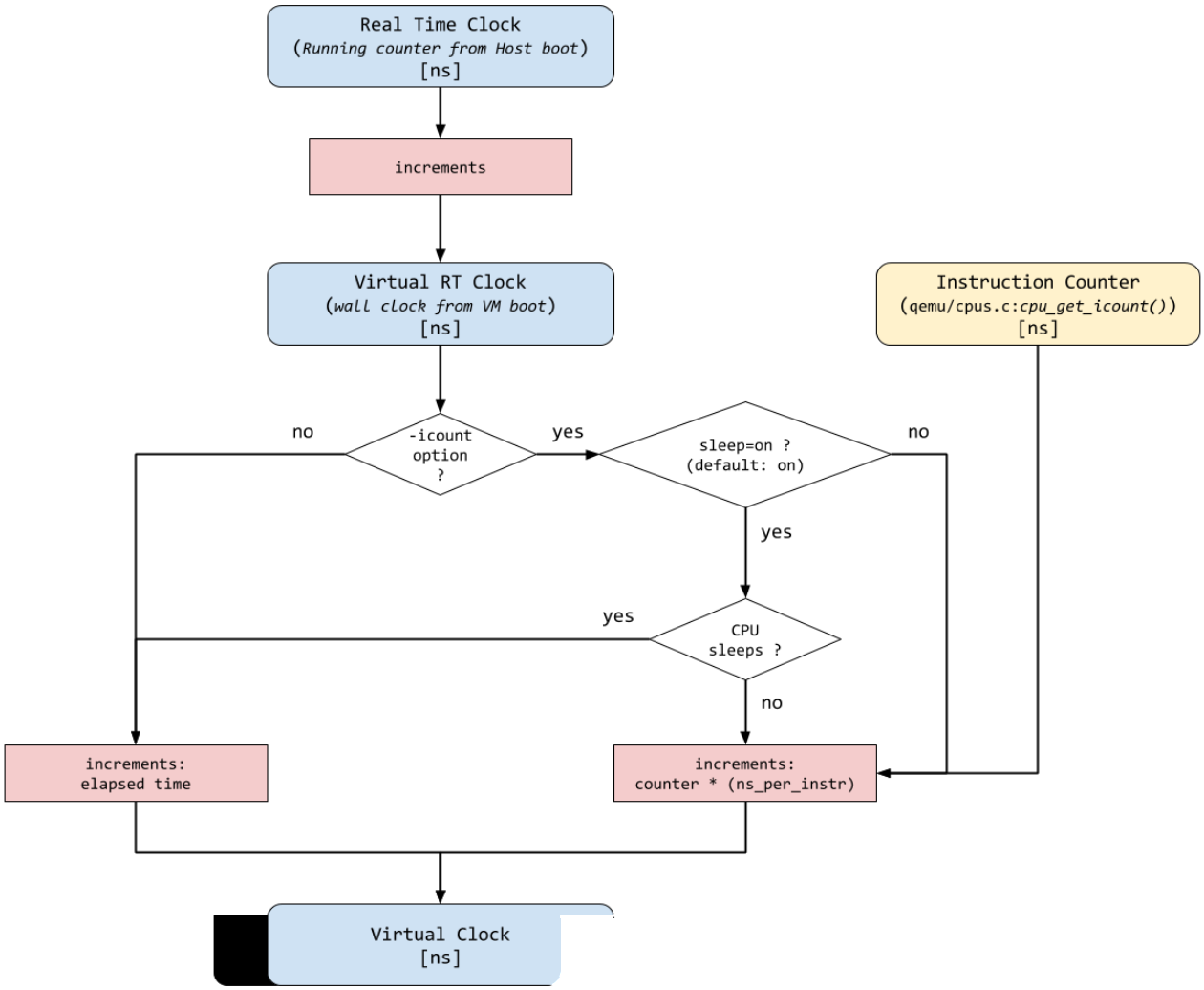


Figure 6.4 - SCHEMA DE LOGIQUE INCREMENTALE DES HORLOGES DE QEMU

PDLT KN J UHQST K est mise à jour via la fonction d b b fd de d d b. Lorsque le mode b est activé, la fonction b fd b met l'horloge à jour en utilisant le compteur d'instructions du processeur virtuel que l'on récupère avec b fd b .

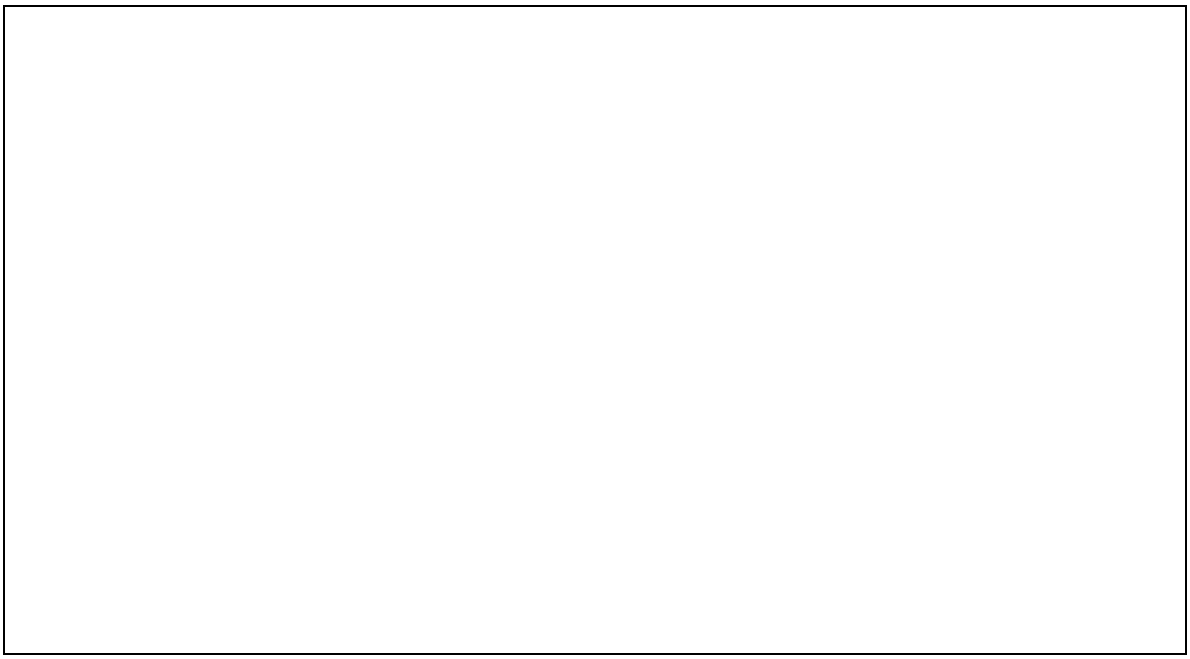
Cependant, si seul le compteur d'instructions du processeur virtuel incrémentait PDLT KN J UHQST K, le temps du système virtuel serait extrêmement ralenti lors de l'inactivité du processeur. C'est la raison d'être de d b a que l'on retrouve dans la chaine d'appel de b fd b sous la forme:

Cette variable globale, définie dans b b, n'est pas mise à jour lorsque le processeur virtuel est occupé, mais seulement lorsqu'il est inactif. Cette compensation s'effectue par une correction de décalage dans la fonction b située dans le fichier b b, appelé par

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

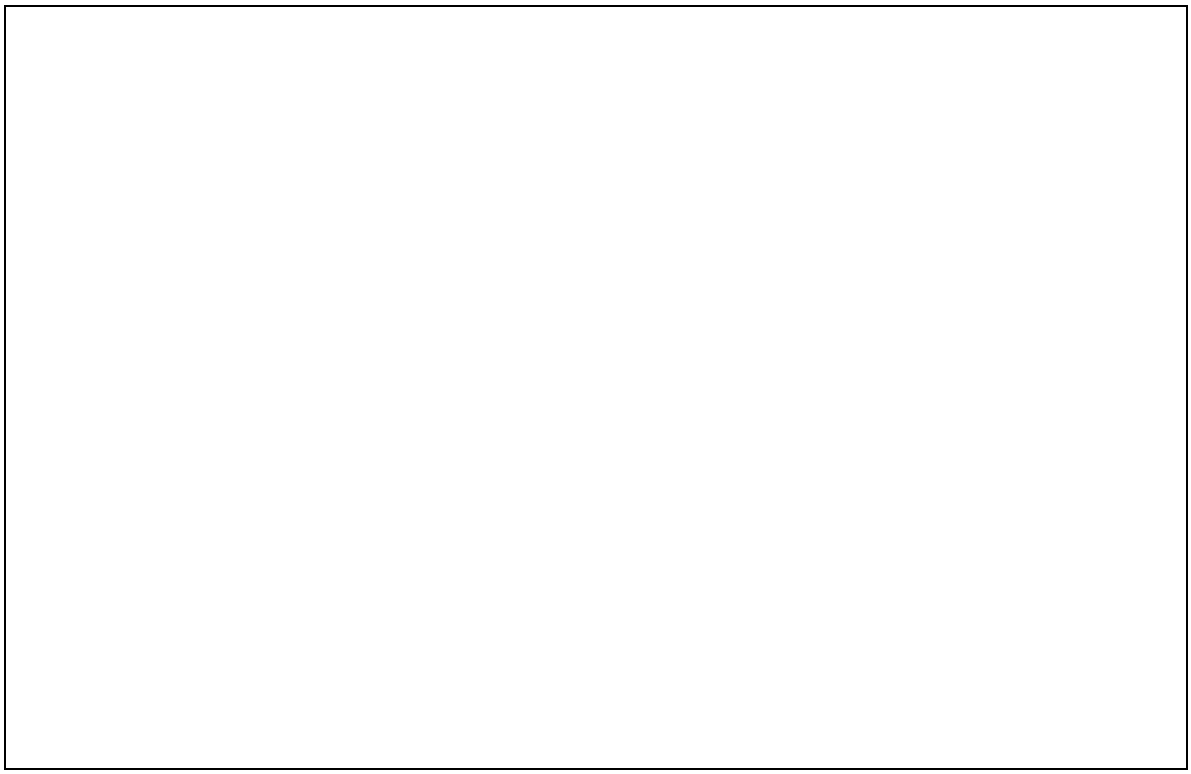
d bb d qui est présent dans la boucle infinie de d bf b d c e dans laquelle s'exécute la simulation des vCPUs en « single-threaded » et dont le fonctionnement est abordé en section 6.1.2.

Comme le montre le code de la fonction d bb d ci-dessous, b est appelé si l'option b est active (variable globale d b), si le ou les vCPUs sont endormis (variable globale b dd) et si la machine virtuelle est active. L'état de la machine virtuelle est récupéré avec la fonction d f (retourne 1 si active sinon retourne 0).



Dans l'extrait de code de la fonction b les parties « thread-safe » ont été supprimées afin de donner plus de visibilité sur la réelle mécanique de correction de l'horloge virtuelle.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.



La fonction b fd b b b dc récupère la valeur du compteur de l'horloge temps réelle de l'hôte et d b a est calculé en fonction du décalage de PDLT KN J QD KSHLD et ainsi que PDLT KN J UHQST K QS. Le détail du calcul de ce décalage est abordé en section 7.2.3

Q q

L'API du système de temps principalement situé dans d d b offre la possibilité de créer des entités nommées des « timers ». Chaque timer appartient à une liste qui est liée à une des horloges de Qemu. Un timer est structure de donnée se situant dans b cd d d . Il représente une valeur de l'horloge associée à sa liste parente à un instant :

La fonction c d b cd c d dans b b et située dans la boucle thread de la fonction d bf b d c e abordée en section 6.1, appelle la fonction d d du fichier d d b se chargeant de mettre une liste de timers à jour.

La création d'un timer s'effectue via la fonction d d . Le premier argument détermine le type d'horloge associé à une liste où Qemu ajoutera le nouveau timer. L'argument ba (ou callback) renseigne un pointeur sur fonction dont le prototype est c PDLTS d c d dont * d sera le troisième argument passé en paramètre au moment de l'appel de fonction. Ce pointeur * d pointe la structure conteneur du timer. Le détail d'utilisation de la callback et est abordé en section 7.2.3. La Figure 6.5 montre que les pointeurs *ba et * d sont tous contenus dans la structure du timer.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

La timer est une valeur future dans le temps par rapport à son l'horloge associée, dans notre cas PDLT KN J UHQST K. Le temps de l'horloge peut être récupéré avec la fonction d b b fd dont un exemple est donné ci-dessous. Une fois que le temps dépasse la valeur du timer, celui expire et appelle la callback.

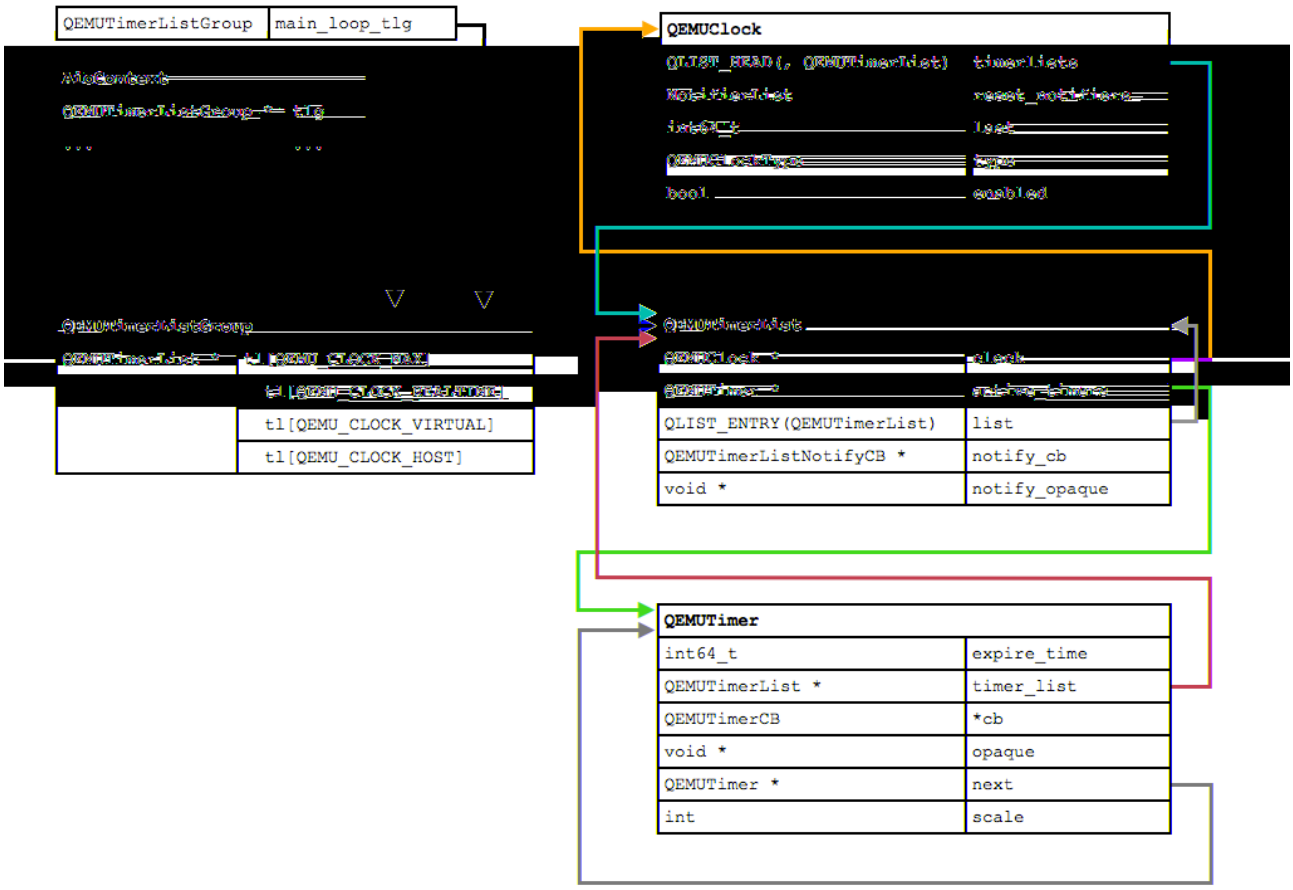


FIGURE 6.5 - SCHEMA DES LIENS ENTRE LES STRUCTURES DE DONNEES DU SYSTEME DE TEMPS DE QEMU

Le réarmement du timer est possible avec la fonction d c ;

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Q

QEMU offre une API pouvant émuler le comportement d'un pin générique GPIO. Les deux modules périphériques développés pendant le stage utilisent cette API. Les sections 7.2.3 et 7.2.4 traitent de la connexion des pins d'interruptions de ces modules avec le contrôleur d'interruption et le contrôleur de routage. La lecture des commits du répertoire projet d sitúes sur les branches de développement et .

Cette section aborde le mécanisme d'initialisation, connexion et du déclanchement d'une interruption entre deux modules périphériques et présente les fonctions de l'API utilisées pour effectuer ces différentes actions.

On assume le terme « pin » que QEMU nomme dans son API d .

La Figure 6.6 présente deux des structures de données principales à la base de l'API. Chaque module périphérique dispose d'une liste chaînée de type M dcFOHNK . Un nœud possède un tableau de pins de type d sur lequel le champ pointe dont la taille du tableau est donnée par soit le champ ou en fonction de si les pins du tableau sont destinées en « input » ou en « output ».

Seules les pointeurs des pins en output (ou « pin-out ») sont stockés dans la structure de données héritant de Cd bdR d définissant l'état du périphérique et abordée en 7.2.3. Stockés dans cette structure données, on peut avoir accès aux pins que l'on donne en paramètre de la fonction d'initialisation c cd f dc Cd bdR d cd d b b d étant le nombre de pins en dessous du pointeur .

Etant de type d même si une pin-out possède un pointeur de fonction c d comme champ, elle ne l'utilise pas à l'inverse d'une pin-in dont le champ c d pointe vers la fonction en charge de l'action signal reçu par une pin-out. Les pin-in sont initialisés avec la fonction c cd f dc Cd bdR d cd d c d c d b b d , étant le nombre de pin que l'on souhaite créer. On peut remarquer qu'aucun pointeur de pin n'est renseigné en argument du prototype de fonction et celui retourne c. QEMU gère les pin-in au sein des nœuds M dcFOHNK indépendamment du code du module périphérique.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

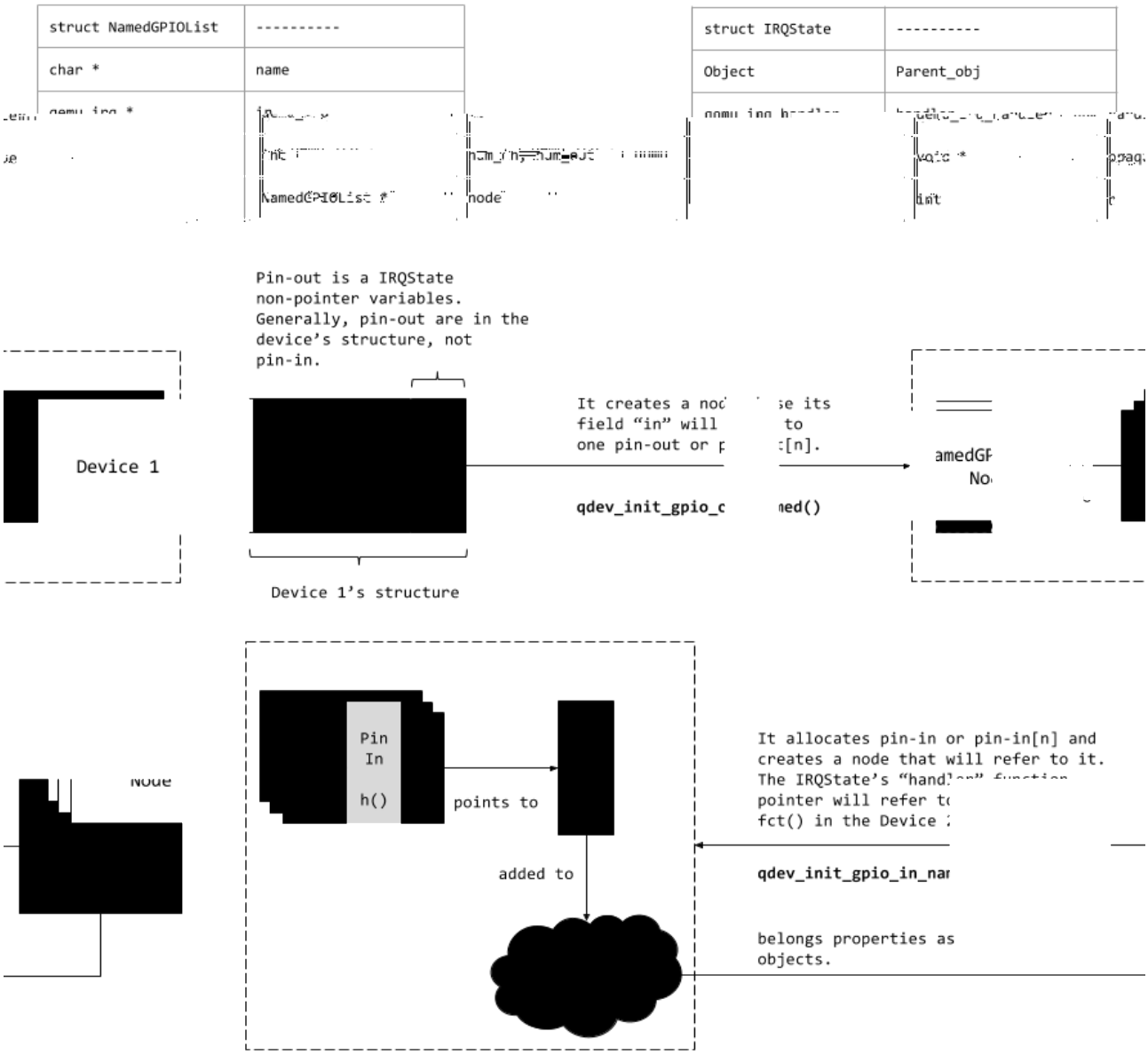


FIGURE 6.6 - SCHEMA D'INITIALISATION DES PINS D'ENTREE ET DE SORTIE

Les pin-out sont donc les seules que l'on verra apparaitre dans le code du module périphérique. A travers un pin-out, un module périphérique 1 déclenchera une fonction du module 2 faisant office de signal.

La connexion entre deux pins de deux modules périphériques s'effectue avec la fonction `cd b db f dc Cd bdR d cd b b d d`. L'argument `cd` correspond au *Device1* en Figure 6.7, `d` correspond au nom

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

donné au nœud M dcF OHNK renseigné lors de l'initialisation des pins. Ce nœud appartient au *Device1* et contient un tableau de pin-out. L'argument est l'index de la pin-out à sélectionner dans le tableau.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

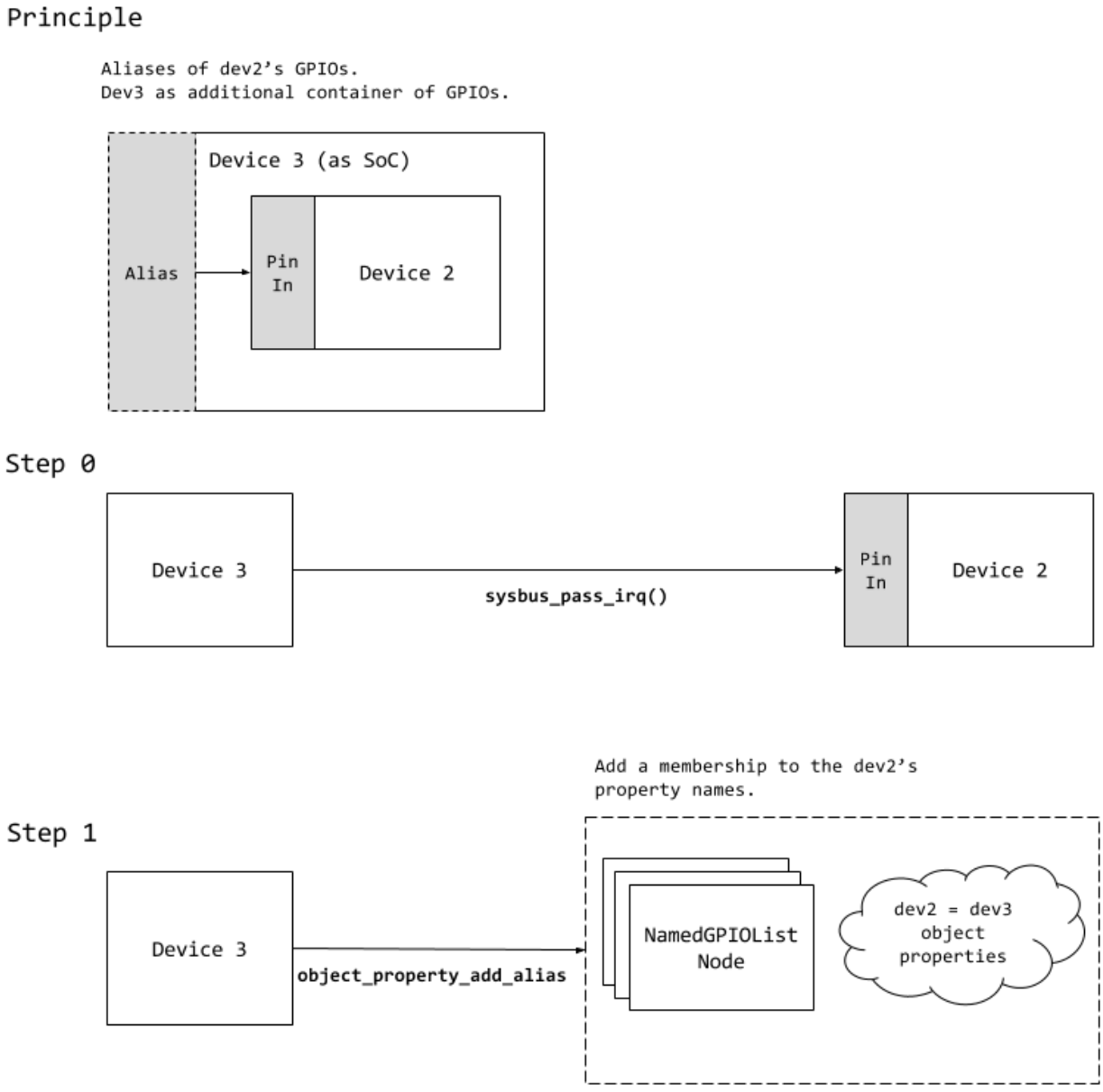


FIGURE 6.8 - SCHEMA DU MECANISME D'ALIAS DES GPIOS

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

K

Un module de QEMU peut être soit un module dit « machine » se référençant avec l'API comme une plateforme qui intègre et assure une cohésion de fonctionnement d'un ensemble de modules dit « périphériques spécifiques» (abrégé en « périphérique »). Les modules périphériques peuvent être uniquement liés à une plateforme ou partagés par plusieurs.

q

Même si il suffit de lire les commits d'un ou plusieurs modules sur la mainline de QEMU pour comprendre et copier leur intégration, il est nécessaire de clarifier certains aspects non-triviaux mais nécessaires à la compréhension générale du processus.

Dans chaque sous dossier de du dossier , un fichier L de d a est présent et permet l'ajout des différents modules machines et périphériques au processus de construction de QEMU.

Plusieurs extraits des fichiers L de d a de l'arborescence sont présentés ci-dessous et vont servir d'exemple pour comprendre le cas d'intégration du matériel de la Raspberry Pi, un module machine qui possède des modules périphériques propres liés à sa plateforme. Ce lien est tel que si on ne souhaite pas construire l'exécutable avec cette plateforme, les modules périphériques associés ne seront pas construits non plus. Dans cette logique de lien plateforme/périphérique, on aborde l'intégration d'un module périphérique partagé par de nombreuses plateformes et donc qui n'est pas proprement lié à un module machine à la construction (ex : le périphérique UART b b). On remarquera que les sous-dossiers portant le nom d'architecture comme ne sont pas mentionnés dans les L de d a car sont par défaut ajoutés à la construction suivant la cible à construire. De cette façon les sources du dossier fd seront compilées.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

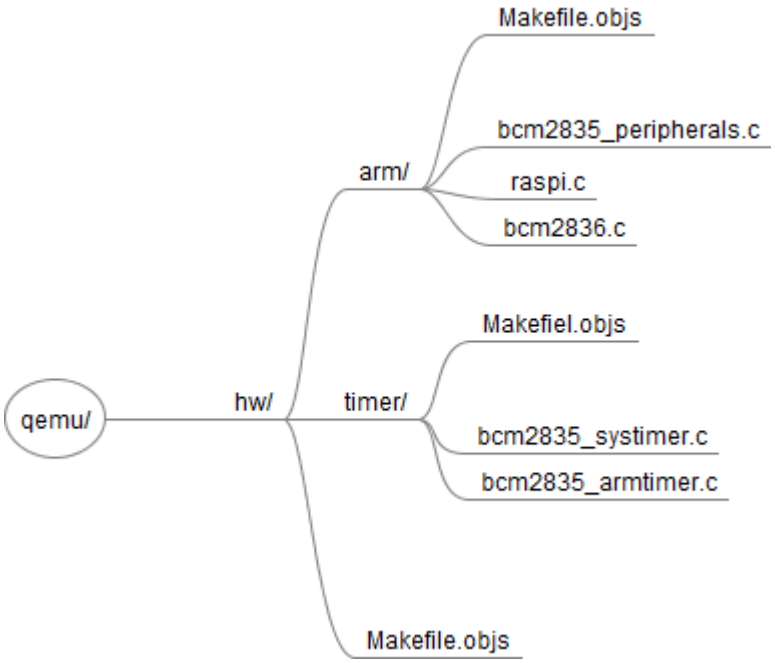


FIGURE 7.1 - ARBORESCENCE MAKEFILE.OBJS

Le fichier L de d a situé dans montre de par les assignations à la variable cd bd c l'ajout des sous-dossiers correspondant aux différents types de périphériques. La variable NMEHF RNESLLT majoritairement utilisée permet la compilation de la plus part des périphériques émulés dans le cas d'une construction de l'exécutable QEMU pour une cible Q G e sélectionnée avec la commande suivante b e f d fd Q G e .

```
hw/Makefile.objs :
-----
cd bd c NMEHF RNESLLT d
```

En descendant dans l'arborescence au matériel destiné à l'architecture ARM situé dans le dossier , on trouve un L de d a dont un extrait est fourni ci-dessous. Le code de la plateforme Raspberry Pi fut correctement découpé en plusieurs parties qui sont proprement liées à celle-ci. Si l'on souhaite que l'ensemble des sources internes au dossier ou dans tout autre dossier périphérique soit lié à la plateforme, il convient d'écrire la variable a de la façon suivante a NMEHF Q ROH . De cette manière, si NMEHF Q ROH , les fichiers objets assignés seront compilés sinon si NMEHF Q RO , ils ne le seront pas. La variable a est utilisée par le processus de construction avec pour lister les sources à compiler et donc les fichiers assignés à a sont ignorés.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

```
hw/arm/Makefile.objs :
-----
 a      NMEHF  Q ROH      ab         d     d         ab
```

Ci-dessous, on présente un extrait illustrant l'association des sources de modules périphériques situés dans d à la plateforme Raspberry Pi par le même jeu de liaison avec la variable a de la forme a NMEHF Q ROH .

```
hw/timer/Makefile.objs :
-----
 a      NMEHF  Q ROH      ab              d     ab              d
```

Comme détaillé dans la section 4.2.2, le fichier de configuration cde b e f e listant le matériel à construire pour la cible e inclut la ligne NMEHF Q ROH . Modifier la valeur de NMEHF Q ROH permet de construire sans les sources de la plateforme Raspberry Pi.

Le module périphérique UART PL011 est dit « partagé ». La suppression de tout module machine au processus de construction n'empêche pas la construction de ses sources. Exactement comme un module machine, le périphérique partagé possède sa propre variable de la forme NMEHF CDUH D M LD afin qu'on puisse le supprimer de la compilation, indépendamment du reste des sources. C'est pour cette raison que la variable b a existe.

```
hw/char/Makefile.objs :
-----
b        a      NMEHF  OK
```

Comme a , elle liste l'ensemble des sources à compiler avec comme traitement spécial que le module périphérique ne sera pas référencé comme un plateforme aux yeux de l'exécutable mais comme un périphérique que l'on peut rajouter à une plateforme par option (en fonction de l'implémentation du module périphérique).

q q

La création d'un module périphérique ou module machine commence par l'analyse d'un des modules de QEMU à prendre comme modèle et base d'implémentation. Dans la section précédente, l'arborescence de l'émulation HW est présentée. Il suffit de la parcourir, comparer les différents modules pour se rendre compte de la structuration d'un module et s'y référer pour toute nouvelle implémentation.

La méthodologie d'analyse plateforme HW et de son implémentation avec quelque uns de ses périphériques sont couvertes par l'article d'Yvan Roch 0, on met donc l'accent sur la partie structurelle d'un module périphérique puis d'un module machine pour la compréhension générale de l'API et du son fonctionnement.

Un module périphérique se compose de 4 parties primaires :

- Une structure de données dédiée à l'implémentation propre d'un composant HW.
- Les fonctions read/write reproduisant le comportement d'un composant HW lors d'opérations de lecture/écriture dans ses registres mémoires.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

- Les fonctions d'initialisation/release permettant d'initialiser les champs de la structure de données dédiée, effectuer des interconnexions avec d'autres composants et attribuer une zone mémoire pour le module en question.
- L'enregistrement du module périphérique (référencement auprès de QEMU).

La Figure 7.2 présente les parties primaires et secondaires qui constituent un module périphérique.

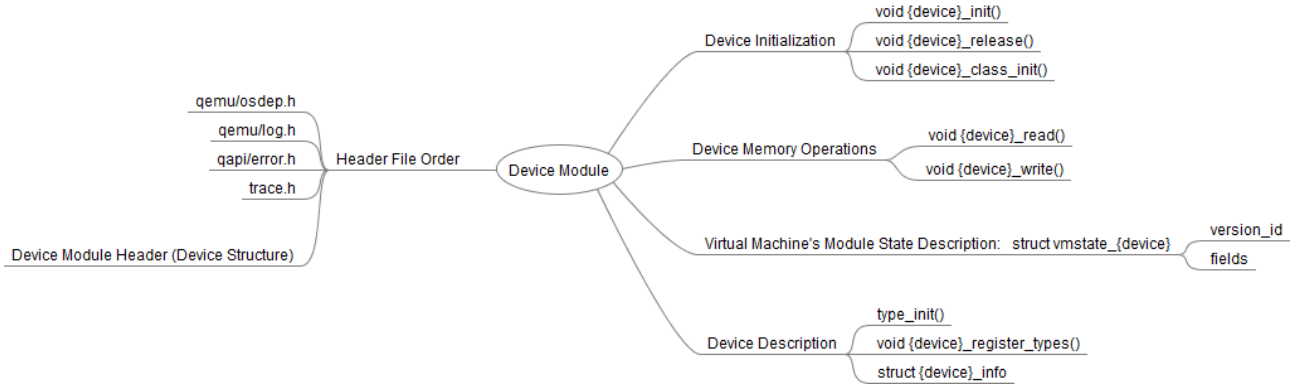
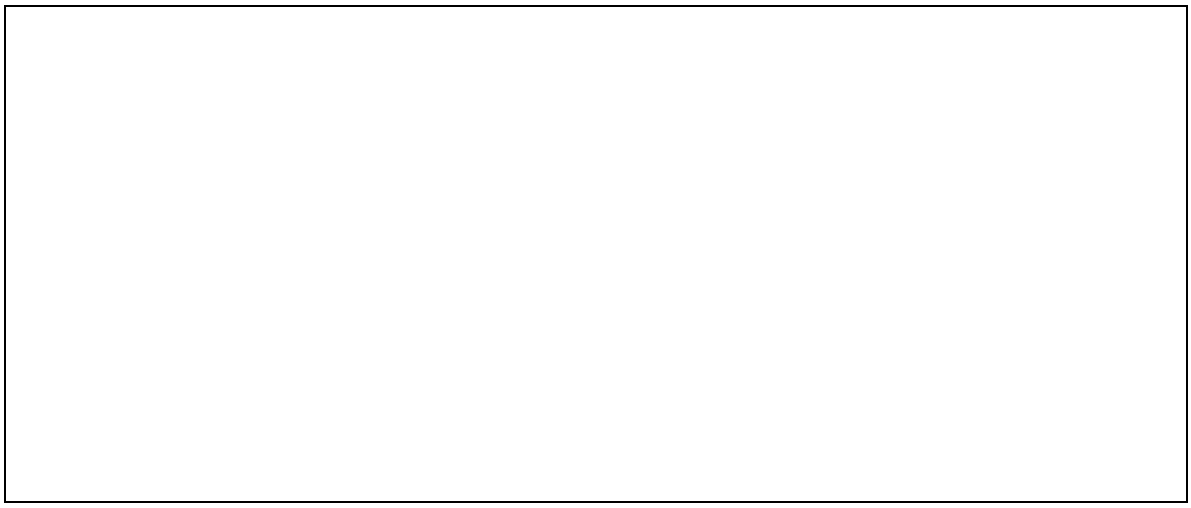


FIGURE 7.2 - QEMU DEVICE MODULE DEFINITION MAPPING

On débute par l'analyse de la structure de données dédiée à l'implémentation du composant HW. Si l'on souhaite rajouter un composant « timer » à QEMU, on se place alors dans le dossier d dédié pour ce type de composant. On crée un ensuite un fichier cd bd b portant un nom en cohérence avec le périphérique. L'ajout de sa structure peut soit se faire dans le fichier source ou dans un fichier d'entête cd bd que l'on place dans le dossier b cd d .

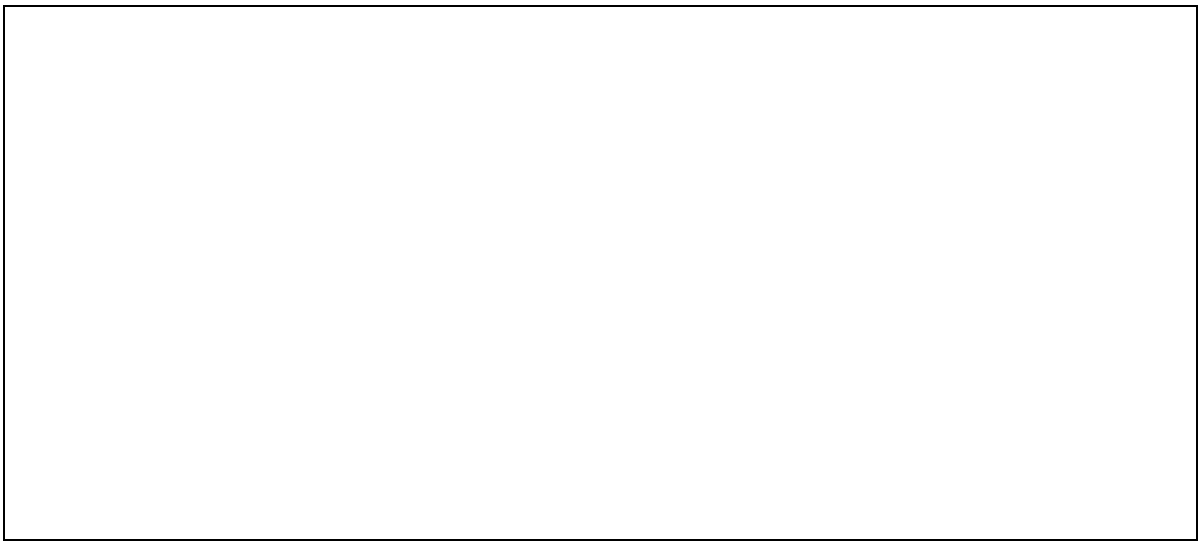
Un exemple générique de structure est donné ci-dessous. Le champ a de type R Cd bd doit toujours être situé en premier. Le standard C certifiant la position fixe du premier champ d'une structure, QEMU s'en sert pour faire de la Programmation Orientée Objets en utilisant le premier champ pour le mécanisme d'héritage de classes (ou structure). Ce champ tel qu'écrit dans l'exemple est à copier dans toutes structures d'implémentations. Son nom peut varier d'un module à l'autre suivant les préférences des développeurs qui utilisent majoritairement d a ou a .

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.



Le second champ d de type Ld Qdf est un pointeur vers une structure dédiée à la zone mémoire du périphérique mappée en mémoire physique et dont l'initialisation s'effectue dans la fonction cd bd . La section 6.2.2 couvre le fonctionnement des différentes zones mémoires de QEMU.

Le reste des champs est propre à l'implémentation du composant et représente généralement les valeurs de ses registres et autres variables utiles à son émulation.



Le fichier b sélection et initialisation de la machine.

Un module machine se compose de 3 parties primaires :

- une structure de données propre à l'implémentation de la machine

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

- une fonction d'initialisation dédiée à la définition de la machine.
- l'enregistrement de la machine auprès de QEMU.

La Figure 7.3 présente les parties primaires et secondaires constituant un module machine.

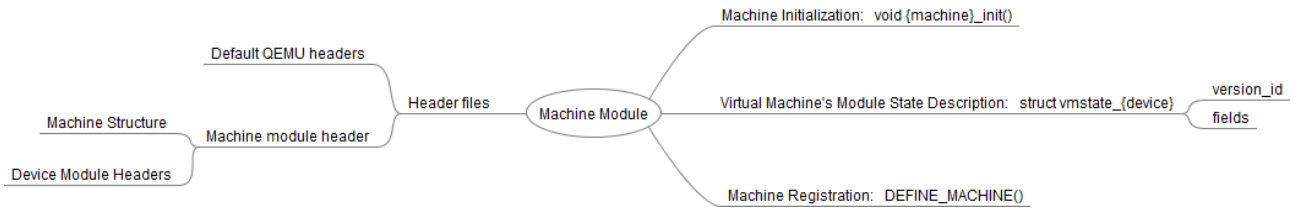


FIGURE 7.3 - QEMU MACHINE MODULE DEFINITION MAPPING

L'exemple ci-dessous présente la définition de la machine dans son fonction d'initialisation.

Q

q

Les différentes relectures et compréhensions du manuel de référence des périphériques du SoC BCM2835, la documentation ARM QA7 et du manuel de référence de l'architecture ARMV7-A ont donné lieu au dessin du schéma de la pour disposer d'une vue claire et concise des différents timers et compteurs associés à leur horloge. Il est actuellement en cours de validation auprès des concepteurs du SoC.

La liste les horloges principales (sans énumérer les PPLs dont la liste exhaustive se trouve dans le dossier c b a b des sources du dépôt d fd) et timers.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

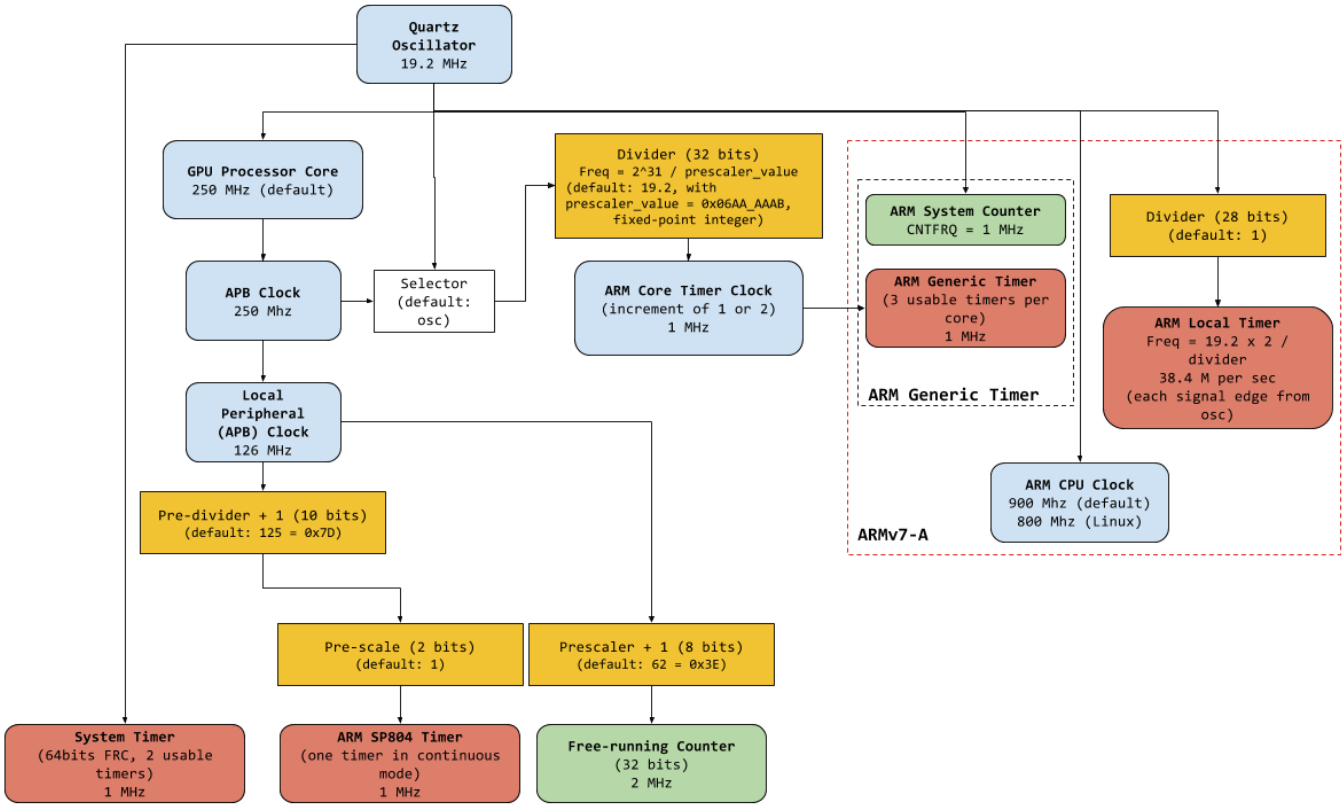


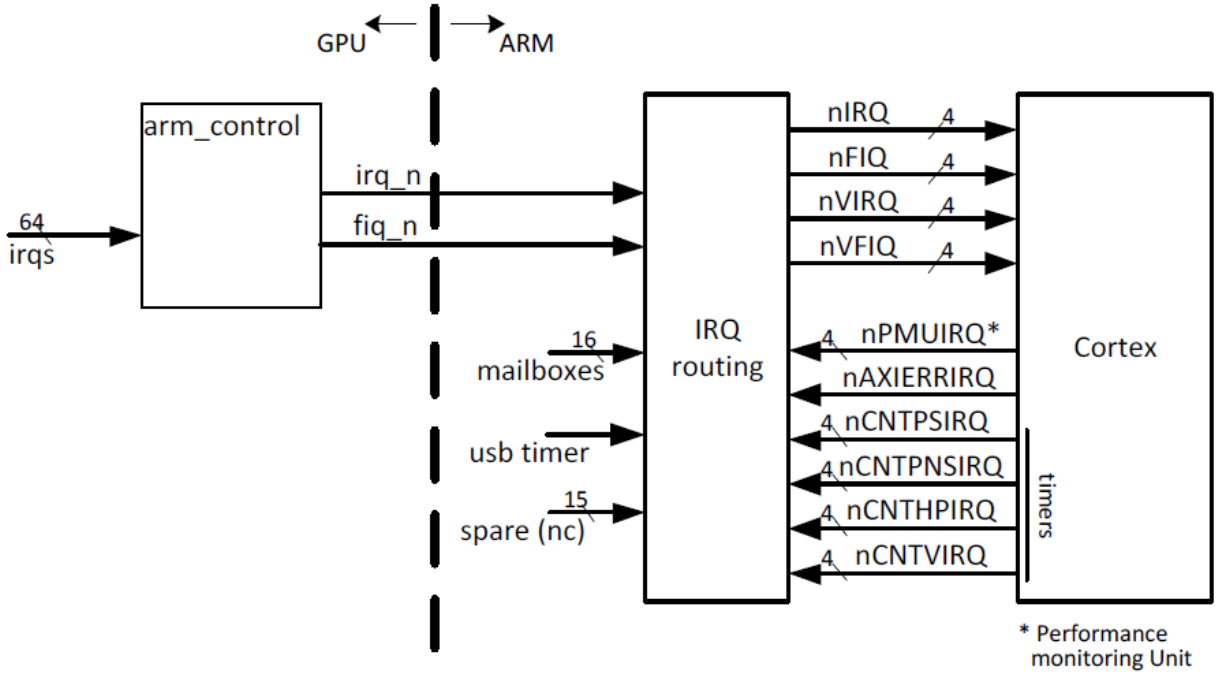
FIGURE 7.4 - ARBRE DES HORLOGES PRINCIPALES, TIMERS ET COMPTEURS DE LA RASPBERRY PI 2 V1.2 ET PI 3

En Figure 7.4, le quartz et horloges sont légendées en bleu, les timers en rouges, les prescalers en jaune et les compteurs libres en vert. La délimitation en pointillés rouges désigne les éléments faisant partie intégrante du processeur ARM et le reste appartenant au SoC BCM2835.

Hors contexte mais néanmoins intéressant de préciser que la fréquence de l'horloge du cœur GPU ainsi que celle du processeur ARM peuvent être modifiées [R12] par l'édition du fichier `b e f` résidant dans la première partition `E S` de la carte SD.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

q



© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

q q

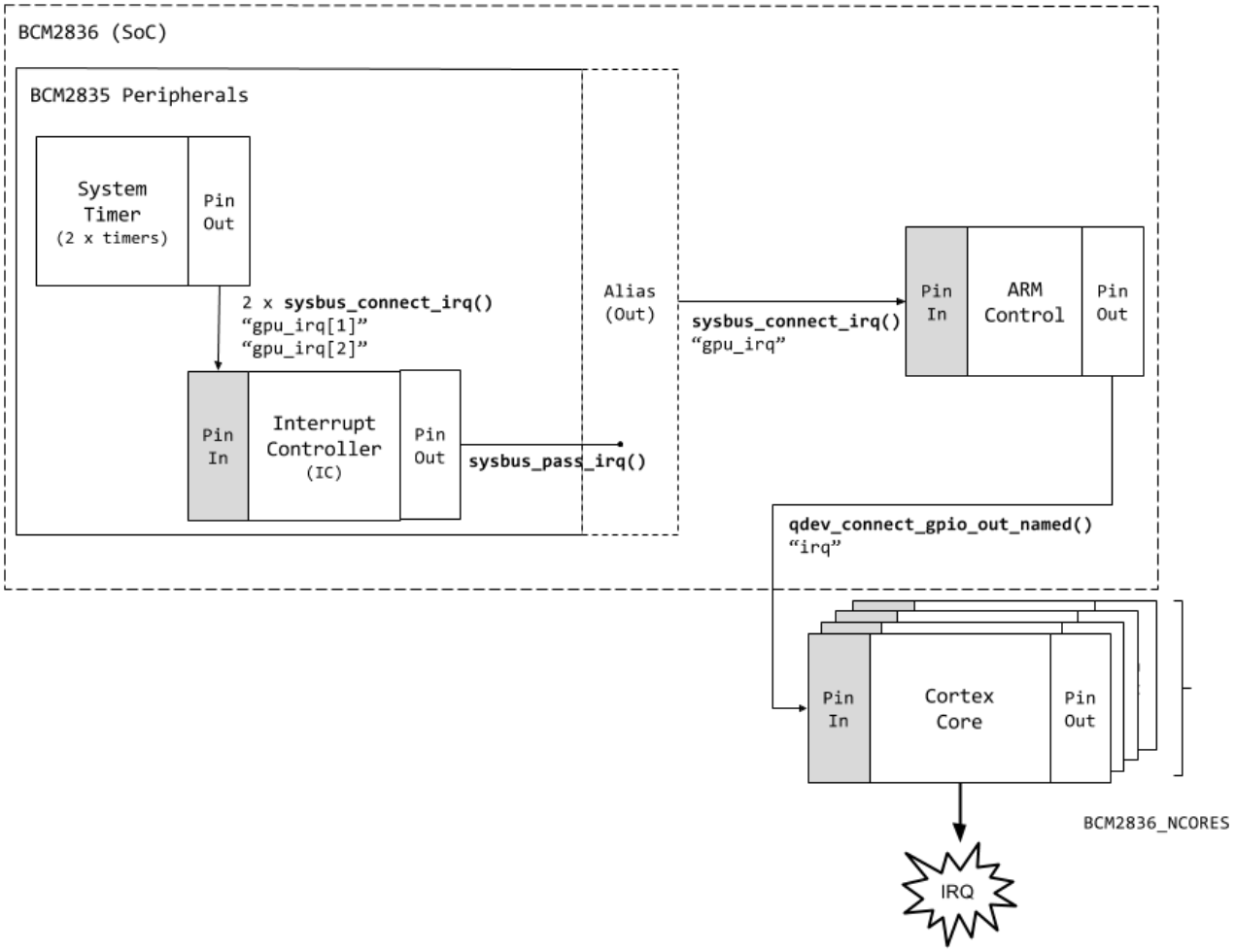
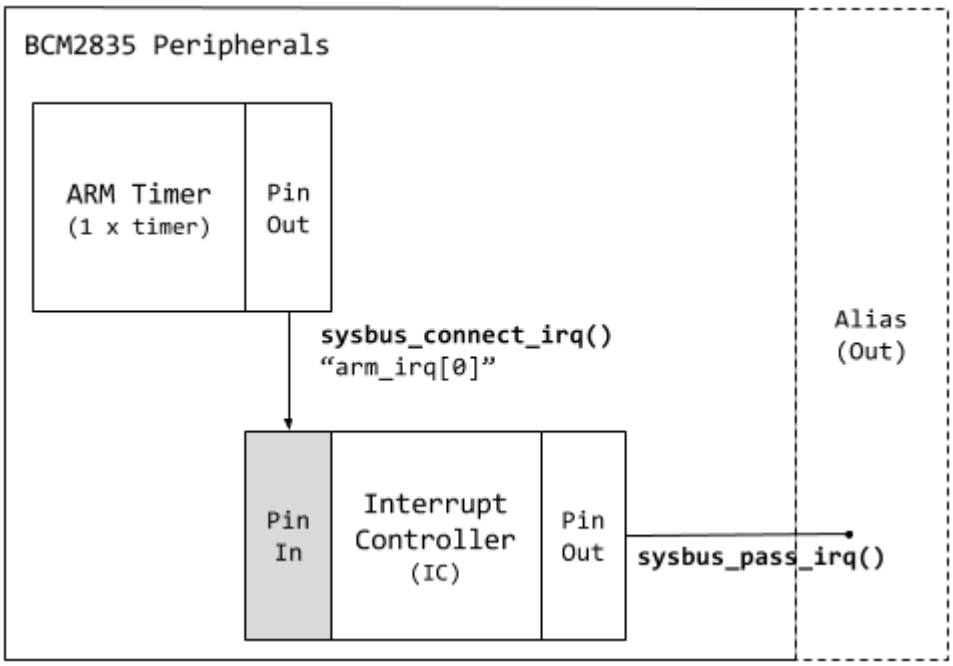


FIGURE 7.5 - SCHEMA DE BRANCHEMENT DES PINS (OU IRQS) DES PERIPHERIQUES IMPLIQUES DANS LE FONCTIONNEMENT DU PERIPHERIQUE BCM2835 SYSTEM TIMER"

q q

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.



Q

Partie DRAFT 5 prévu pour le 07/08.

Q

Partie DRAFT 5 prévu pour le 07/08.

Partie DRAFT 5 prévu pour le 07/08.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Q

Partie DRAFT 5 prévu pour le 07/08.

Q

Partie DRAFT 5 prévu pour le 07/08.

Partie DRAFT 5 prévu pour le 07/08.

Partie DRAFT 5 prévu pour le 07/08.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Pour les deux cartes électroniques, la liste des composants émulés fut obtenue à partir des commandes *monitor* de QEMU ainsi que du code source des deux modules machines.

A1.1. QQ

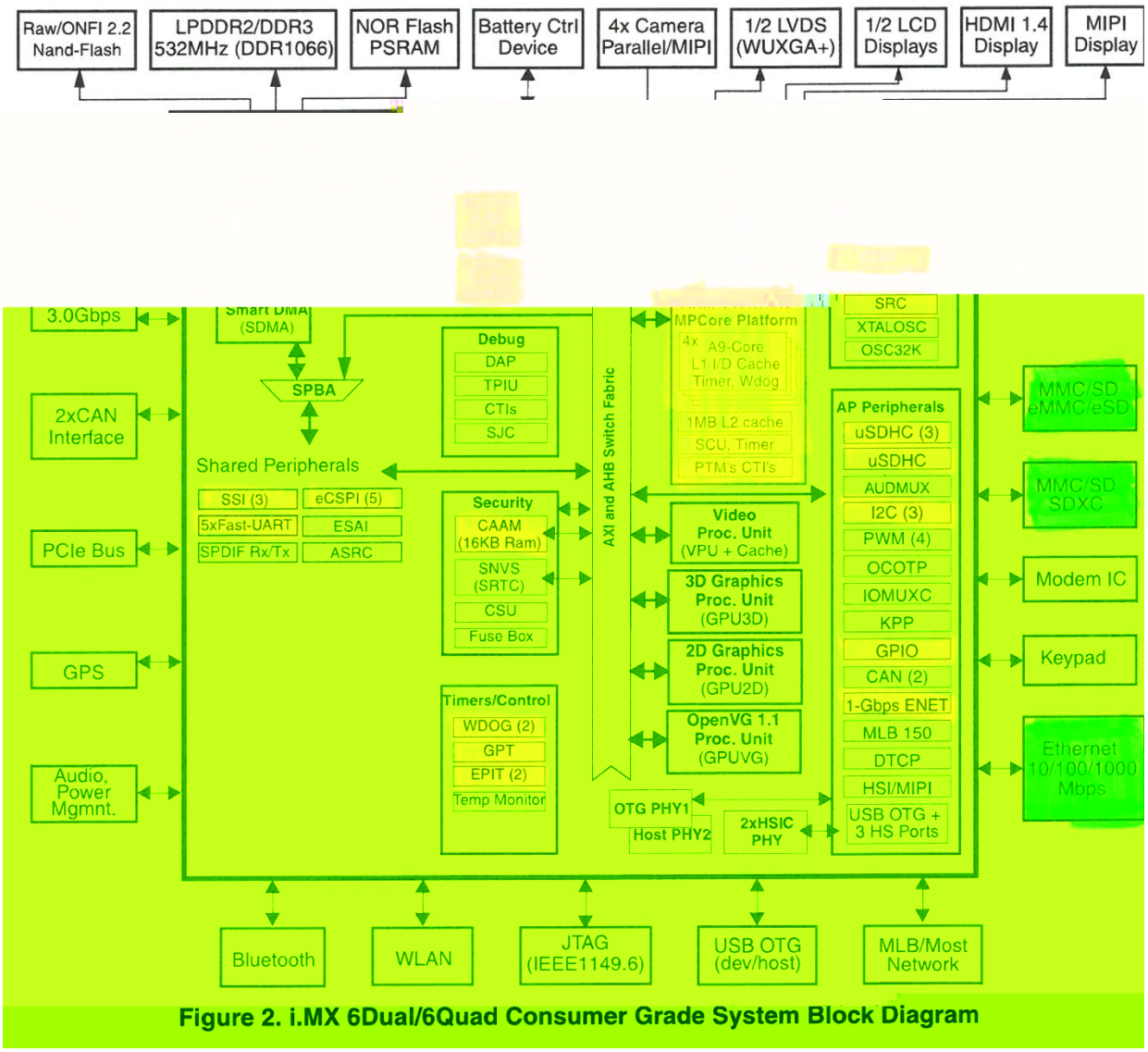


FIGURE 8.1 - I.MX6 EMULATED BLOCKS UNDER QEMU

QEMU émule les modules ci-dessus en

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

A1.2. QQ Q Q

L'émulation de la Raspberry Pi 2 inclut :

- ARM Processeur Cortex-A15 x 4
- BCM2835 SoC (voir Tableau A1.1)
- Contrôleur d'interruption ARM CPU (IRQ/FIQ)

Q Q	q
System Timer	-
Contrôleur d'interruptions	X
GPIO	X
PCM / I2S	-
I2C	-
SPI	-
PWM	X
Contrôleur DMA	X
UART	X
USB (Registers & SMSC LAN9512)	-
Module RNG	X
SDHCI / EMMC	X

TABLEAU A1.1 - LISTE DES PERIPHERIQUES DU SOC BCM2835 EMULE DANS QEMU

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

Page laissée intentionnellement blanche.

© Propriété de Safran Electronics & Defense - Reproduction et divulgation interdites/Disclosure and copy prohibited.

POWERED BY TRUST

18-20 quai du Point du Jour
92659 Boulogne-Billancourt Cedex
Tél. : + 33 (0)1 55 60 38 00
Fax : + 33 (0)1 55 60 38 95
Société par actions simplifiée au capital de 372.884.426,40 €
480 107 911 R.C.S. NANTERRE