



# Jekyll theme for documentation — mydoc product

version 5.0

*Last generated: February 05, 2017*

---

© 2016 Your company. This is a boilerplate copyright statement... All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

# Table of Contents

## Overview

Get started .....	3
-------------------	---

## Conception et exploration d'architectures, multi-coeurs, réseaux sur puces

Cours .....	4
TP .....	19

## Conférences technologiques

Notes .....	20
-------------	----

## Droit et informatique

Notes .....	48
-------------	----

## Etude de cas d'implantation d'un SLE

Projet .....	49
--------------	----

## Modélisation transactionnelle des systèmes sur puces

Cours .....	50
TP .....	58

## Systèmes distribués

Cours .....	59
TP .....	68

## Sécurité des systèmes embarqués

Cours .....	69
TP .....	77

## Systèmes temps-réel, modélisation réaliste et implantation multi-taches

Cours .....	78
TP .....	79

## Tolérance aux fautes

Cours..... 80

**Validation des systèmes embarqués**

Cours..... 81

TP ..... 88

# Page de présentation

**Summary:** Ce site permet de rédiger des notes de cours en Markdown et de retrouver l'ensemble des notes sur un site plutôt bien organisé hébergé gratuitement sur Github.

## Présentation

Ce site regroupe l'ensemble de mes prises de notes lors de nos cours de 3ème années à l'ENSIMAG en filière Systèmes et Logiciels Embarqués.

La particularité de ce site est qu'il est entièrement rédigé en Markdown, et hébergé gratuitement sur github.

## Contribution

Il est tout à fait possible de contribuer à l'élaboration de ce site ou de son contenu en ajoutant vos notes aux miennes.

Pour cela, n'hésitez pas à me contacter, il vous suffira de cloner le dépôt, et de faire des merge request.

## Organisation

Chaque cours apparaît dans le panneau latéral à gauche de l'écran.

Pour chaque cours, apparaît 1 ou 2 catégorie(s) : Les cours, les notes ou les TP.

Le contenu de chaque catégorie se trouve dans le fichier :

`pages/mydoc/initial\_du\_cour/nom\_de\_catégorie.md`

[]:

# Notes de Cours

Partiel possible (<https://ensiwiki.ensimag.fr/images/6/64/SLE-SOC-exam-2012.pdf>)

## Cours du 03/10/2016

- SoC :
  - Processeur + Cache
  - Bloc mem
  - Contrôleur Mémoire externe (RAM)
  - Bloc de calcul pour soulager/accélérer les calculs
  - Bloc Analogique
  - Bus
- Bus : Débit + Protocole
- RAM : < 10Mo de Cache + 4 à 128 Go de cache
- CPU : Unité de calcul
- DSP : -> contrôle temps réel
  - - Puissance de calcul
  - - Programmation complexe
- ASIC/IP :
  - - Puissance de calcul, traitement réguliers, consommation
  - - Rigidité, Temps de dév, Peu adapté à un contrôle complexe
- Analogique :
  - - Faible conso, Grande intégration
  - - Bruit, Forte dépendance à la techno
- MEMS : MCNC
- SoC : Assemblage de plusieurs techno
- Comment réduire la consommation d'énergie :

- Conso =>  $P = 1/2 * C * V^2 * f$ 
  - ASIC
  - RAM : SRAM / DRAM
  - Techno :
    - Fils moins long
    - Petit, densité
    - Compromis en taille des transistors(performance) et consommation
  - Archi du système :
    - Bus -> longueur de fils -> charge du circuit diminu
    - Paralléliser -> on peut diminuer la fréquence et diminuer la tension
  - Tension : Lié à la techno
  - Amélioration combinatoire -> éviter les glitches
  - Simplification du SW

## Cours du 10/10/16

lien du cours

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCEAMC/document/SLE\\_SoC\\_infrastructure.pdf?cidReq=ENSIMAG5MMCEAMC&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCEAMC/document/SLE_SoC_infrastructure.pdf?cidReq=ENSIMAG5MMCEAMC&id_session=0&gidReq=0&origin=))

## Bus

✓ **Tip:** Un bus système sert à assurer le transfère de données entre les organes d'un SoC

❗ **Note:** Si on veut utiliser 2 CPU et 1 mémoire avec un bus  
Dans ce cas, l'instruction n'est plus fetch en 1 cycle (plus forcement)

- Connectique:
  - Bus adresse
  - Bus donnée
  - Bus de contrôle

- Physique:
  - Protocole d'échange

Comment gérer plusieurs maître et plusieurs esclave ?

### **Bus Trois Etat**

- Point positif:
  - Grande modularité
  - Connectivité maître-esclave simple
  - Observabilité simple
- Point négatif:
  - Fréquence de fonctionnement faible à cause de la capacité des 3 états
  - Test complexe
  - Grande longueur des connexions
  - Layout délicat: Antenne

## Contrôleur de bus

✓ **Tip:** Le contrôleur de bus répond aux requêtes des maîtres. Il donne accès au bus à une demande selon  
Une politique de priorité  
Une gestion temporelle du bus

Exemple d'arbitre pour un bus :

Arbitre de bus

*Exemple d'arbitre*

Conclusion du petit exercice: on est limité en nombre de maître et d'esclave.

La solution consiste à *hiérarchiser les bus*

- Conversion de protocole par un Bridge
  - +: pas cher, pas de matériel
  - -: peut vraiment bloquer en terme de performance si plusieurs maître veulent communiquer en même temps
- Stockage intermédiaire dans une SRAM



- +: Le maitre écrit dans la RAM, et se préoccupe plus de rien, libère le bus
- -: Cher

## DMA : Direct Memory Address

✓ **Tip:** DMA = C'est un automate de transfert automatique de données d'un périphérique vers la mémoire

Burst: plutôt que de faire une requête et d'attendre la donnée après plusieurs cycles d'attente, on fait une demande de n données d'un coup.

Caractéristique du bus:

- Nombre de canaux
- Gestion du bus
- Transferts
  - Groupé : Burst
  - Auto-incrémenté
  - Canaux chaînés
- Mode de transfert
  - FIFO
  - Taille de données
- Interruption
  - Fin de transfert
  - Erreur
- Priorité des canaux

## Exemple de bus

*CoreConnect*

Proposé par IBM

# Mémoire

Lien des slides

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCEAMC/document/SoC\\_memoire.pdf?cidReq=ENSIMAG5MMCEAMC&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCEAMC/document/SoC_memoire.pdf?cidReq=ENSIMAG5MMCEAMC&id_session=0&gidReq=0&origin=))

✓ **Tip:** Les mémoires sont des tableaux de points mémoire assemblés en ligne de mots

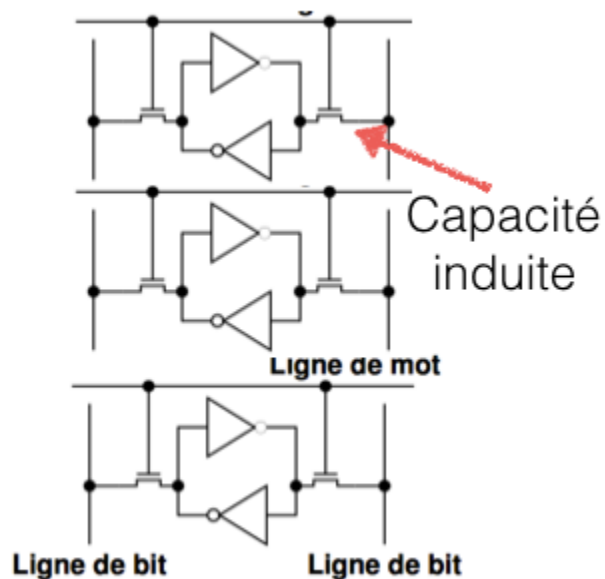
C'est la technologie des points mémoires qui détermine les performances d'une mémoire.

- Densité
- Rapidité

Décomposition de mémoire en bloc car c'est impossible d'avoir un gros seul bloc  
Sinon elles seraient trop lentes

## Cours du 17/10/16

La capacité induite des lignes d'information fait que l'on ne peut pas utiliser le même modèle de mémoire si elles sont trop grandes.



### *schéma d'un plan mémoire*

#### SRAM

1 point mémoire c'est 6 transistors

C'est donc rapide et un techno standard.

Mais de faible densité, et non synthétisable.

#### SDRAM : Synchronus Dynamic RAM

C'est un transistor et une capacité:

Points positifs:

- Grande densité car petite
- Pas chère

Points négatifs:

- Perd la valeur à chaque lecture car le condensateur se décharge.
- A cause des fuites de courant, le condensateur se décharge, le niveau passe sous le niveau 1.

Il faut donc faire constamment des lectures et re-écriture pour mémoriser les données.

Les SDRAM doivent être composés de plusieurs petites mémoires

#### Résumé

- Densité élevée
- Débit élevé
- Coût faible
- Grande latence
- Technologie spécifique
- Consommation: pas terrible car il faut bcp lire et re-écrire
- Non synthétisable

#### ROM

Petit et rapide mais en lecture seul

## Flash

Non volatile, conserve les données hors tension et accès sécurisé mais lent et techno non standard.

## Registre

16 transistors par point mémoire, très chère et gros mais rapide

## Les SPRAM

Une Scratch-Pad RAM (SPRAM) est une mémoire directement connectée à un coeur de processeur. Elle est visible dans l'espace mémoire.

Il est possible d'y accéder en un cycle d'horloge, sans arbitrage. Une SPRAM doit être gérée par le logiciel. Les données peuvent être lues/écrites par:

- Les instructions LD/ST standards
- Un DMA interne au processeur

## Block-RAM : BRAM

Une Block-RAM (BRAM) est une mémoire accessible par le bus système. C'est une IP esclave accédée par les IPs maîtres.

En général, les BRAM servent aux échanges de données entre les processeurs et IPs maîtres.

## Caches

Les cache (antémémoire) sont des mémoires rapides qui disposent d'un mécanisme automatique de copie/écriture des données/instructions de la mémoire principale.

Exploite les principes de localité spatiale et temporelle.

Les principaux paramètres d'efficacité sont: - La taille mémoire - La politique de gestion

Le principal critère d'efficacité est le taux de défaut de cache, qui dépend de ces paramètres et de l'application.

Temps moyen d'accès = Temps hit + Taux de défaut \* Pénalité de défaut

## Cours du 07/11/16

### Etude de cas

#### lien des slides du cours

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCEAMC/document/multimedia.pdf?cidReq=ENSIMAG5MMCEAMC&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCEAMC/document/multimedia.pdf?cidReq=ENSIMAG5MMCEAMC&id_session=0&gidReq=0&origin=))

## Cours du 28/11/17

Cours de petrot, début du cours d'archi des machines modernes [Lien du site du cours \(https://ensiwiki.ensimag.fr/index.php/Transparents\\_du\\_cours\\_d%27Archi\\_3A\)](https://ensiwiki.ensimag.fr/index.php/Transparents_du_cours_d%27Archi_3A)

## Super pipeline vs superscalar

Super pipeline = plein de petit étage

Superscalar = plusieurs instructions en parallèle

Exemple du MIMPS Superscalar:

2 pipelines en parallèle (pas 2 identiques)

Une partie fait les opérations de type :

- opération entre registres
- opération entre registres et constante (immédiat)

Une autre partie fait :

- Opération d'accès à la mémoire

Or, 1 instruction sur 5 est un accès mémoire, donc pas forcément bien répartie

Pour qu'un superscalar puisse exécuter 4 instructions dans le même cycle, il ne faut pas que les 4 instructions soient dépendantes.

- Dans un processeur VLIW, le compilateur devra délivrer des groupes de 4 instructions indépendantes
- Dans un processeur 'in-order superscalar', c'est l'HW qui va exécuter au maximum 4 instructions si elles sont indépendantes. Sinon, insert des stall
- Dans un processeur 'out-of order superscalar', c'est l'HW qui choisi 4

instructions indépendante qu'il pourrait exécuter mtn.

Exemple slide : 6

Avec processeur MIMPS classique : 7 cycles

In order : 5 cycles

Cycle	Pipeline NOIRE	Pipeline JAUNE
1	...	lw \$6
2	add \$5	...
3	sub \$9	lw \$7
4	add \$3	sw \$5
5	add \$11	...

Out of order : 4 cycles

Cycle	Pipeline NOIRE	Pipeline JAUNE
1	sub \$9	lw \$6
2	add \$5	...
3	sub \$3	lw \$7
4	add \$11	sw \$5

Execution time = Number of instructions \* CPI \* cycle time

## ILP : Instruction Level Parallism

Pipeline CPI = Ideal pipeline CPI + Structural stalls (= stall after lw) + data hazard stall (= for data which result of prior instruction, tjs dans le pipeline) + control stalls (= par exemple, les instructions exectuter pour les branchements)

## Dépendances

### Data dépendance

```
I: add r1, r2, r3  
J: sub r4, r1, r5
```

l'instr J est **data dépendant** de l'instr I

### Anti-dépendance

```
I: sub r4, r1, r3  
J: add r1, r2, r3  
K: mul r6, r1, r7
```

J est anti-dépendant à I, ce n'est pas une vraie dépendance, dépendance de nom

### Output dépendance

```
I: sub r1, r4, r3  
J: add r1, r2, r3  
K: omul r6, r1, r7
```

J est output dépendant car après ces deux instructions la valeur de r1 doit toujours être la même

Pour les dépendances de nom, on pourrait faire utiliser une IP de renommage des registres pour en avoir plus que 32 sans pour autant augmenter la taille des instructions.

## Cours du 05/12/2016

## OUT OF ORDER EXECUTION

Pipeline classique INORDER :

IF	DEC	EXE	MEM	WB
----	-----	-----	-----	----

Dans le pipeline on ajout un étage : RENAMING entre DEC et EXE

IF	DEC	REN	EXE	MEM	WB
----	-----	-----	-----	-----	----

2 tables sont utilisées : le RENAME MAP et le FREE MAP

RENAME MAP = table de correspondance entre registre logiciel et registre physique qui gère les dépendances

FREE MAP = par exemple une fifo de registres libres permettant d'être utilisé pour le renommage

Dans le pipeline on ajout un étage : DISPATCH entre DIS et EXE

IF	DEC	REN	DIS	EXE	MEM	WB
----	-----	-----	-----	-----	-----	----

## Prédiction de branchement

Solution naïve dans MIPS, POWER PC, PENTIUM ...

On a une Branch History Table (BHT) qui est indexé par les 14 bits de poids faible de PC. (taille de BHT =  $2^{14}$  bits)

On mémorise uniquement dans cette table si le branchement a été pris ou non la dernière fois que ce PC a été exécuté.

Cette méthode peut poser des problèmes de collision...

Cette méthode est trop basique car avec une boucle for on a forcément 2 miss prediction ... à la 1ère itération et à la dernière

## Cours du 04/01/17

## Vector processors

Apparus dans les années 80 jusqu'en 90. Puis réapparition avec les GPU.

### Flynn's Taxonomy

Von Neumann Architecture = CPU + Memoire (fetch + exec)

Année 70, M. Flynn distingue plusieurs types d'architectures:



- SISD = Single Instruction, Single Data
- SIMD = Single Instruction, Multiple Data
- MISD = Multiple instruction, Single Data (Ça n'est pas utile, donc n'existe pas)
- MIMD = Multiple instruction, Multiple Data (cas de nos ordinateurs multi-cœur, mais nécessite de la synchro ... )

SIMD : C'est ce qui correspond au GPU ou au machine vectoriel

Utilisation :

Pour faire une addition entre 25 couples d'éléments, j'ai juste à créer deux vecteurs de 25 éléments et de faire 1 seule addition vectoriel de ces 2 vecteurs.

Souvent pour faire les opération vectorielles, il faut de grand pipeline, donc il y a une grande latence, mais aussi un grand débit.

Permet de faire des :

- opérations simple:  $C[i] = A[i] + B[i]$
- opération avec indirection :  $C[i] = A[i] + B[D[i]]$
- opération conditionnelle :  $C[i] = A[i] + B[i]$ , si  $A[i] > 0$  par exemple

### Multimédia Extension (MMX)

Forcement 64 bits, donc 8x8 ou 4x16 ou 2x32

L'utilisation de ces extensions ou des processeurs vectoriels est très spécifiques :

- Video
- Audio
- Image Processing
- Data encryption

## Architecture Very Long Instruction Word (VLIW)

Apparu dans les années 80

Pas mal utilisé pour tout ce qui ne peut pas consommer bcp, les switch router, ...

Instruction de taille 128 bits typiquement

CISC: (x86) RISC: (ARM, powerPC, ...) VLIW: (TI, ST, HP, ...)

Un VLIW c'est un RISC avec des instructions plus grandes et un pipeline plus grand. VLIW = super RISC.

En VLIW : une instruction de 128 bits contient dans les 128 bits plusieurs instructions simples, qui seront exécutées en MÊME TEMPS.

Le HW d'un VLIW est assez simple car le HW ne prend pas du tout en charge les hazard (problème de dépendance entre instructions, etc)

La sémantique de l'asm en VLIW étant différente, il faut connaître la latence de chaque instruction.

## Dynamic Scheduling : Thomazulo

Thomazulo algo slides (<https://ensiwiki.ensimag.fr/images/2/25/02-Tomasulo.pdf>)

Trouver le parallélisme possible entre les instructions à l'exécution. Avec renommage de registre et ordonnancement des instructions.

Le but de faire du Out-of-Order est de

## Cours du 09/01/2017

### Avant 2004

Un système classique (avant 2004) est constitué d'un bus sur lequel vient se connecter tous les autres périphériques

### Après 2004

Intel a arrêté de faire augmenter la fréquence des chips pour démultiplier le nombre de processeur

Si dans un système on démultiplie le nombre de CPU, disons  $n$  CPU.

Chaque CPU a besoin d'une "bande passante"  $b$  pour échanger des données avec la mémoire.

Il faut donc que le bus ait une bande passante possible d'au moins  $n * b$ .

Admettons que notre cache par CPU ait un taux de miss de 10%, et qu'il fait un accès mémoire tous les 5 instructions

Alors il y aura 1 instruction sur 50 qui fait un miss.

Sachant qu'il y a 8 mots par ligne.

De plus, Le CPU tourne à 2 GHz mais le bus tourne qu'à 800MHz

Se que l'on constate c'est que la latence à une forme exponentiel et explose au alentour de 10% = 1 requêtes tous les 10 cycles

## Les NoC

[lien des slides](#)

(<https://ensiwiki.ensimag.fr/images/6/64/SLE-SOC-exam-2012.pd://ensiwiki.ensimag.fr/images/e/e5/NoCS-Tutorial.pdf>)

[lien des slides](#) (<https://ensiwiki.ensimag.fr/images/9/9b/Ginosar.pdf>)

[Partiel corrigé](#) (<https://ensiwiki.ensimag.fr/images/2/23/SLE-SOC-exam-part-fred.pdf>)

Pour remédier à cela, création des NoC

Utiliser des architectures réseaux qui ont de bonnes propriété: par exemple le thor.

### Vocabulaire

- *Message* is a basic communication entity
- *Flit* is a basic flow control unit
- *Phit* is the basic unit of the physical layer
- *Direct Network* each switch connecter to a node
- *Indirect Network* with switch not connected to any node
- *Hop* is the basic communication action from node to switch or from switch to switch
- *Diameter* is the length of the maximum shortest path between any two nodes measured in hops
- *Routing* distance between two nods is the number of hops on a route
- *Average* distance is the average of the routing distance over all pair of nodes

### Basic switching techniques

- Circuit switching : Il existe un chemin entre source et destination, et le message utilise ce chemin.
- Packet switching : Chaque packet connait la destination et chaque packet est envoyé indépendamment dans le réseau.
- Store and forward packet switching : Tous les packets sont mémorisés dans tous les switches.

[]:



# Introduction

## EMPTY

␣:

## Notes pris lors de la conférence

### 07/10/16 : Conception de système embarqués contraint sous RTOS à cloisonnement

Depuis 2008 chez ELSYS Design

- Localisation:
  - USA
  - France
  - Europe
- Clients:
  - Aerospatial
  - Defense
  - Energie
  - Medical
  - Miltimedia
  - ...

Entretien technique, Excellence technique.

#### Système Temps-Réel

Interface plus simple avec le matériel:

- Gestion de taches
- Gestion des irq
- Temps
- Communication
- Ressources

Plusieurs taches qui paraissent s'exécuter en même temps sont en fait exécutés séquentiellement. -> Il faut donc ordonnancer les tâches.

Si c'est multi-coeur, plusieurs tâches peuvent s'exécuter en même temps.

- Concurrence :
  - Priorités différentes, quelle politique ?
    - Run to completion
    - Utilisation de "pthread\_yield"
    - Time Sharing
    - Tick système
- Prémption :
  - Scheduler peut suspendre une tâche
  - Problème de réentrance: interrompre une opération qui modifie une variable en mémoire
  - Synchronisation
- Inversion de priorité :
  - Permet de donner une plus grande priorité à un tâche qui à la ressource.
- Dead Locks :
  - Tjs prendre les ressources dans le même ordre

-> Systèmes temps réels :

Capacité de traiter en un temps déterminé un ensemble d'évènements

- Sans perdre d'évènements
  - Déterminisme de l'ordre de traitement de ces évènements
- Temps-réel ne veut pas forcément dire performant (pas tjs)

### Problématique de la programmation en contexte temps-réel

- Zones critiques Désactivation des interruptions => Attention !!
- Solution
  - Éviter au plus de désactiver les interruptions
- Optimisation du compilateur
  - Plus efficace en activant l'optimisation du compilateur
  - Mais on peut avoir de mauvaise surprise :
    - Debug pas à pas chelou
    - Volatile pour les accès registres

- Volatile pour une variable partagé entre contexte de tâche et contexte d'interruption

### Différent temps réel

- Temps réel “mou” :
  - Les contraintes de temps peuvent de temps en temps ne pas être respecter
  - La performance moyenne est prise en compte
- Temps réel “dur” :
  - doit TJS TJS respecter les contraintes de temps
  - Pire cas est pris en compte

### RT-OS

Temps de réponse garanti:

- Points clés :
  - Algo d'ordonnancement
  - Temps de latence des irq
  - Temps de latence de changement de contexte
- Priorités des tâches et des interruptions
  - Traitant d'irq minimalisé en context d'irq
  - On peut reporter l'irq dans des tâches de haute priorité

### Quelle RT-OS

- Cout
- Fonctionnalité
- Performance temps-réel
- Encombrement
- Offre logiciel
  - Protocoles de com. (TCP/IP)
  - Applications (serveur web,...)
- Outils de développement et mise au point
  - IDE



- Debugger
- Analyseur de temps-réel
- Simulateur

ASP = Architecture Support Package

BSP = Board Support Package

### Comment mettre en oeuvre un tel système

- Cycle en V mais bon ... En théorie :
  - Invalidation des hypothèses de départ
  - Changement des exigences en cours de route
  - Identification d'un problème technique
- Solutions ? Dérisquer
  - Analyse de risques
  - Actions préventives
  - Développement itératif
  - Envisager le pire cas
    - Périodicité des signaux
    - Exactitude du pilotage
    - Freq des irq

### Notion de cloisonnement

Exemple du point de vente:

- IHM
  - OS non temps-réel
  - IHM peut planter, pas grave
- Un module de paiement
  - Sans OS, certifié
  - Module dédié et certifié -> NE DOIT JAMAIS PLANTER

=> Enjeux : Rassembler les deux fonctions sur un seul matériel

- Passage de cloisonnement "matériel" à cloisonnement "logiciel"
- Réduction des coûts

## Type de cloisonnement

- Cloisonnement mémoire
  - Protection d'un espace cloisonné contre les débordements mémoire d'une autre cloison
  - Chaque proc est cloisonnée dans son cloisonnement
  - Utilisation de la MMU du uP pour exposer un espace d'adr virtuel
  - Tous accès interdits lèvera une exception, traité par l'OS
- Cloisonnement temporelle
  - Disponibilité d'une quantité garantie de CPU à une partition temporelle
  - Découper l'exec en 'Time Slices' sur lesquels on sait parfaitement quel tâches peut utiliser les ressources
- OS Classique
  - Mémoire partagé
  - IPC, messagerie, signaux
  - Mutex ...
- OS Cloisonnement
  - Aucun risque de corruption d'une autre cloison
  - Pas de mémoire partagé
  - Pas de communication possible entre les cloisons : COMMENT FAIRE ?

Exemple : Dans INTEGRITY

- Partition "Virtual Address Space"
  - Memory région : mem partagée mappé dans les deux VAS
  - Com bidirectionnelle synchrone ou asynchrone
  - Semaphore : synchro entre les VAS

Du coup en cas de crash ?

- Memory Region : Données corrompues, accès concurrents (RO, WO, semaphore)

- Connection : Corruption des données; Absence de réponse à une requête
- Semaphore : Blocage de la ressource INTEGRITY à mis en place un "Resource Manager"

### Scheduling en env cloisonné

Quelle prio pour chaque cloison ? Les cloisons doivent respecter les exigences RT  
Que faire avec les taches de même priorité mais de cloisons différentes ?

- CHEZ INTEGRITY:
  - Prioritized
    - Tache e plus haute priorité et qui est prête, obtient le CPU
  - Preemptive
  - Enhanced Partition Scheduler
    - La tâche de plus haute priorité de la partition temporelle courante obtient le CPU.

## 14/10/16 : Matlab: Polyspace

### Introduction

Polyspace depuis 2010 dans la vérification de code

3500 salarié 900 SW eng 90 products

Polyspace Dev Team

- 30 eng
- Paris + Montbonot
- Aéronautique
- Automobile

Interprétation abstraite créé par M et Mme Cousot au 3ème étage de l'ENSIMAG  
[wiki \(https://fr.wikipedia.org/wiki/Interprétation\\_abstraite\)](https://fr.wikipedia.org/wiki/Interprétation_abstraite)

Outil d'analyse abstraite = Théorie qui prend en entrée un programme qui va de manière automatique va prouver qu'il n'y aura pas d'erreurs à l'exécution du programme

Programme = Ensemble de C/C++ qui compile

Automatique = L'utilisateur ne dois pas intervenir

Prouver = se baser sur une base mathématique solide

Réponse = Elle doit être 'OUI', 'NON' et 'Peut être', le dernier choix est OBLIGATOIRE (c'est prouvé)  
 A l'exécution = Quand le programme s'exécutera en condition réelle.

### Exemple

- Ariane 5: Overflow
- Missile patriote: Erreur de float de 10-4% -> 137 mètres
- Panne électricité au USA: data race
- Volvo crash: capteur accumule de l'imprécision

### Les bases de l'interprétation abstraite

Pas la peine de tout connaître dans un programme, il suffit de connaître parfaitement les informations que l'on a besoin

Grille d'interprétation de signe par exemple

Top				
<=	>=0			
<0	=0	>0		
ottom				B

On joue alors le programme non plus en essayant de savoir TOUT mais utilisant un élément dans la grille Les propriétés que l'on utilise peut être divers:

- Signe
- Intervals
- Octagons
- Polyhedre
- Disjonctions

### Enjeux techniques

- Programme avec pointeur
- Programme avec structures
- Tableau
- Complexe control flow

- C++

Exemple avec des pointeurs:

- Erreur de déréférencement de pointeurs
  - Pour chaque déréférencement, il faut vérifier si le pointeur est valide

Calcul d'alias = Ensemble de variables qui ont des relations entre elles ex :

$$\begin{array}{l} P \rightarrow X \\ \quad \searrow \\ \quad \rightarrow Y \end{array}$$

Signifie que P peu pointer vers X ou Y et c'est tout

Contexte insensitive : ne différencie pas les contextes d'appels de fonction :  
résultat assez large

Contexte sensitive : prend en compte les contextes d'appels de fonctions, plus stricte

Flow sensitive : Idem avec les if etc

Flow insensitive :

Le sensitive est trop couteux ... donc en pratique on fait du insensitive et on essaye de raffiner

Algorithme de Anderson:

- écrire les points-to (lien de pointeurs) = calcul d'alias
- Il faut poser les équations.
- Il faut résoudre une équation de point fixe.
- Complexité en  $n^3$ .

Algorithme de Steensgaard:

- Quand on fait le graph des alias, on le fait plus qu'une flèche qui part d'un pointeur, vers une boite contenant toutes les possibilités.

Algo de Steensgaard

*Exemple d'algo de Steensgaard*

- Cette modification permet de réduire la complexité de  $n^3$  à  $n^1$
- Mais cette méthode ne permet pas de manipuler des tableaux ou des structures

Technique maison de chez MathWorks:

- Vue de graph imbriqué
- Pour les tableaux, on peut mémoriser que le pointeurs est dans une case sans savoir la quelle

Pour prendre en compte les tableaux et les structures l'aglo se complexifie : de  $n^1$  à  $n^2$

### Enjeux technologiques

Il faut que le produit soit intéressant, simple, efficace à utiliser.

Pour réussir, on se pose les questions suivantes:

- Que va se service du produit ?
  - 1) SW dev
  - 2) SW eng
  - 3) Quality eng
- Chaque utilisateur ont des utilisations différentes
  - 1) Facile à utiliser, quelques clics, pas contraignant.
    - Prendre en compte le fait que le code testé ne sera pas forcément entièrement présent, ou pas tout doit être testé
    - Différent code généré par des compilateurs exotiques
    - Faire des interprétations dans les modèles
    - Intégration dans un IDE, git etc
  - 2) Utilisable de façon efficace et conviviale
    - Interval pour variable, points-to pour les pointeurs, relations entre variable
    - Code non atteignable
    - Pour chaque RTE, OUI, NON, PEUT-ETRE
    - Prise en compte de bcp de fautes "classiques" : Overflow, NULL, Not Initialized, etc ...
    - Coloration
      - Vert = C'est SUR que le code est safe.
      - Rouge = C'est SUR que le code génère une erreur à l'exécution.
      - Gris = Code mort

- Orange = Ce morceau de code PEU poser problème (PEUT-ÊTRE...)
- Violet = Violation de règle de codage : MISRA-C/C++
- Information au passage de souris: intervalle d'une variable etc
- 3) Traiter les résultats de façon pas trop fastidieuse
  - On peut guider l'annalyseur en lui forçant la main au vert sur certain point
  - Tools
  - Métrique de la qualité d'un code
- Comment faire pour que l'outil fonctionne pour de vrais cas, qui ont beaucoup de LoC
  - Exemple : 200,000 LoC -> la plus longue chaine d'alias peut être de 16122 case mémoire; 5536 set d'alias
  - Pour réduire la complexité, on peut faire du SSA [Static Single Assignment](https://fr.wikipedia.org/wiki/Static_single_assignment_form)  
([https://fr.wikipedia.org/wiki/Static\\_single\\_assignment\\_form](https://fr.wikipedia.org/wiki/Static_single_assignment_form))

## Conclusion

Conférence POPL/PLDL = Meilleur conférence en info théorique

en 2016 -> 35 papier sur les traitements de pointeurs

De plus en plus de floatant -> NaN, infinity, subnormals -> Problèmes complexes

On peut pas avoir tout vert, il faudra forcément passer par des oranges

Anecdote :

Livre sur le C++

# 21/10/2016 : Marine National: Information quantique, 40 ans après ...

*Il y a forcément eu des progrès techniques qui ont permis le passage en pratique :*

- Progrès technique
- Miniaturisation: Loi de Moore
- Progrès industriel: Production de masse

## Statut historique

### Évolution de la pensée: du corpuscule au quantique

#### *Max Planck et la radiation des corps noir*

- Change de couleur
- $T = f(\nu)$

#### *Albert Einstein : l'effet photo-électrique*

- Généralisation à la lumière
- Effet photovoltaïque et photo-électrique
- Contradiction des lois classiques de la thermodynamique car il y a un seuil : fonction de travail =  $\theta$  =  $h\nu$

#### *Bohr : modèle atomique*

- Quantification des niveaux atomiques
- Spectre d'émission pour passer d'un niveau à un autre, émission d'un photo

#### *Trous d'Young*

- Généralisation aux particules ( $\gamma$ ,  $e^-$ )
- Interférences constructives et destructive
- Dualité onde corpuscule ( $e^-$  dans les fentes d'Young)

#### *Erwin Schrödinger*

- L'atome est une onde, son état est une fonction d'onde qui est présente dans l'Equation de Schrödinger
- Probabilité ( $t$ ,  $r$ )

## Mécanique quantique: Notions fondamentales

#### *Interprétation de Copenhague*

- Un système est décrit par une fonction d'onde



- Cette fonction d'onde est décrite par l'équation de Schrödinger
- Probabilité (en mécanique quantique, on parle tjs de proba), mesure (La mesure détruit des choses) et décohérence
- Principe d'incertitude  $\Delta x \cdot \Delta p > \hbar/2$  et  $\Delta E \cdot \Delta t > \hbar/2$ , on ne peut pas connaître la vitesse d'une particule ET sa position, idem avec temps et énergie

### *Intrication/Superposition quantique*

- État de superposition

### *Chat de Schrödinger, paradoxe d'EPR*

- L'état du chat est en superposition :  
( $|\text{DEAD}\rangle$  or  $|\text{ALIVE}\rangle$ ) OR  $(1/\sqrt{2})(|\text{DEAD}\rangle + |\text{ALIVE}\rangle)$
- Rôle de l'observateur, de la mesure

### *Décohérence et cohérence*

- Temps de cohérence, temps maximum durant lequel l'objet est en superposition, durant le quel on peut profiter de l'effet quantique

### *Mesure faible*

- Comment mesurer sans détruire ?
- On mesure faiblement, sans trop changer l'état du système, mais un grand nombre de fois pour établir des statistiques

## **Développement de l'information quantique**

Combinaison de facteurs

### *Théorie CLASSIQUE de l'informatio: Shannon*

- Source, réception, bruit
- Nyquist, 1928, Quantité d'information que l'on peut transporter
- Hartley, 1929, Qualité de l'information
- Shannon, 1948, Regroupement + approfondissement  
Même si on perd des informations, si le taux est < à la capacité de transmission, avec des codes correcteurs d'erreur, on peut retrouver ~100%

### *Les nanotechnologies: Feynman*

- Physicien électrique: projet Manhattan et Challenger
- 1er transistor en 1947
- 1er circuit intégré 1958
- 1959: Possibilité de manipuler et de créer des objets nanométriques
  - Concepte de nano-robots, nano-manipulateur

### *Miniaturisation des transistores*

- Loi de Moore, loi purement économique
- Dimension atomique: plus on se rapproche de la taille atomique plus il y a des problèmes :  $E_c$  devient  $> k_B T$
- Intégration 3D : Connections ?, chauffage ? Efficacité ? Beaucoup de nouveaux problèmes
- Problèmes calculatoires: Dans l'espace de Hilbert on arrive très très vite à des besoins de ressources gigantesque

### *L'implantation mono-atomique*

- Réussir à implanter 1 seul atome

### *Concept de machine : Turing*

- Automatisation des tâches
  - Contraignant
  - Calculs plus complexes, nombreux
  - Rapidités et qualités des décisions
- Machine de Turing (virtuelle)
  - Code, état initial , courant, etc

### *Concept du qubit*

- Etat classique
  - 0 ou 1 - > le bit
- Etat quantique

- 0 ou 1 ou 1+0 ou 0+1 -> le Qubit
- Niveau atomique

### Opération

- Les opérations sont des mouvements dans l'espace

### Concept de clonage et téléportation

- Non clonage
  - Pas de copies identique d'un état quantique inconnue
  - Seul les états originaux sont possible
  - Pas de techniques classique de corrections d'erreurs
  - Copies imparfaites
  - En effet, copier c'est d'abords mesuré = détruire en quantique
- Pas de téléportation, pas de transmission (superluminale)
  - Pas de mesure précise possible (part d'incertitude)
  - Pas de reconstruction d'état quantiques via des états classiques

### Nécessite d'un contrôle

- Circuit adapté de mesure pour la quantique
  - Electronique basses températures
  - Mesure classique
  - Archi complexe
- Les mesures elles même doivent être différentes
  - réflectométrie radio fréquence
  - Ampli trans impédance

### Status actuel

### Industrie vs. Académique

- Deux point de vue différent
  - **Industrie:** L'industrie veut que ca soit peu couteux, simple, extensif et compatible avec leur production industrielle

- **Recherche:** On cherche l'intérêt de la recherche
- Avantage / inconvénients:
  - Pression des lobbys (D-Wave en 2011, "Quantum annealing", pas de superposition, pas de temps de cohérence)
  - Pas forcément le plus intéressant, facile ou le mieux scientifiquement

### Différente approches et qbits

- Ces technos mélange plusieurs support d'information : photo, e- etc

### *Condition de (Bruce) Kane pour faire PRATIQUEMENT un Qubit*

- Conditions de bases :
  - Définir le qubit
  - Initialiser le système
  - Déterminer un ensemble d'opérations universelles
  - Avoir un temps de cohérence long
  - Lire des information avec de grande probabilité
  - Réaliser un grand nombre de qubits
- Conditions délocalisées
  - Propagés
  - ???

### *Modèle de Qubit de Kane*

- Structure MOS en Si
- Spin nucléaire (mémoire), électronique (qubit)

### *Qubit semiconducteur*

#### Boites quantiques

- Qubit de charge (Temps de cohérence = 100uS): Deux places accécible à un e-, si il est là alors on a un 1 sinon on a un 0.
- Qubit de spin (Temps = 1à5ms et jusque 100ms dans Si) : spin + -> 1 spin - -> 0
- Implantation mono ou bi-atomique / STM: N = 1-2, T = 45s, T refroidi > 1h

### *Qubit supraconducteur*

- Molécule  $T = 3\text{ms}$
- Jonction Josephson  $T = 2\mu\text{s}$
- Diamant qui a des défauts avec les NV
- Piège à ions : confiner des atomes grâce à des lasers

### *Le graphène et autres mono couches*

Découvert en 2004 à Manchester

### *Les matériaux topologiques*

- Les isolants topologiques sont isolants en leur cœur et conducteurs en surface.
- Si on le coupe, on recrée des surfaces, il redevient conducteur sur toutes ces surfaces
- Longueur de cohérence = 300 à 600 nm à 300mK
- Peut être utilisé pour faire des Bus ou des Trains quantiques

### *Qubit volants*

- Avec des photons, pour les satellites ?

### *Interaction localisé - délocalisé*

### *Projet et financement*

Australie: subventionné par US Army

Europe : Pays-Bas, UK

USA

Canada

Chine

Russie

Japon

## Applications

### *basé sur l'intrication des états*

- Calculs parallèles, vitesse accrue/ puce classique
  - Pas de transmission de données car 2 éléments, même distant, connaissent les mêmes informations
  - Gestion de trafic
  - Médecine
  - Astronomie
  - Etude du génome humain
- Algorithme de Shor: factorisation des grands nombres
  - Classique réduction : calcul du PGCD
  - Quantique : accélération
  - $(\log N)^3$  au lieu de  $\exp(\log(N)^{1/3})$

### *basé sur l'effondrement de la fonction d'onde par le phénomène de mesure*

- Inviolabilité, cryptographie
  - Transmissions financière sécurisées (civil)
  - Réseaux ultra sécurisés (DARPA, militaire)
  - Transmission par satellite (Canada-Israël)

### *Codage et chiffage classique*

- Classique : pour que ça soit plus dur, on augmente le nombre de bits
- Quantique : va pouvoir aller bcp plus vite, il faut trouver d'autre moyen pour crypter de l'information

### *Supériorité vs. Infériorité*

- Rapport de forces entre les nations

## Conclusions

Cohérence : les temps de cohérence ne sont plus vraiment un problème aujourd'hui, les T sont de plus en plus grand

Passage à l'échelle : Dépend des approches, mais en Si oui pq pas

Déplacement de l'information: Comment faire communiquer les ordinateurs quantique entre eux ?

- Fibre optique, mais sa qualité est des fois un problème
- Comment faire des Répétiteurs : Clonage ???
- Bus de Qubits ??

Inviolabilité: beaucoup de recherche sur le bruit quantique et les mesures faibles

Aspect stratégique pour la défense :

- Modernisation, automatisation et intégration des nanotechnologies ou des réseaux rendent les systèmes très complexe
- Comment maîtriser cette modernisation et la protéger ces systèmes

## 16/12/16 : BH Technologie, IoT, Jérôme DEGRYSE

Le partenaire des villes intelligentes

Start up, depuis 20 ans -> PME 45 personnes

9M€ de CA, 50% de R&D

Les clients sont les villes

Présence Européenne

- Optimisation de l'éclairage publique: Aider les villes à optimiser l'éclairage des villes
- Aide les villes à la récolte des déchets

## Optimisation de l'éclairage

- Gestion du temps de l'éclairage
- Gestion de la puissance en fonction de l'heure de la journée
- Relève d'incident sur les lignes
- Cloud de supervision
  - Connaître tous les informations des armoires de distribution
  - Envoyer des alertes en cas de problèmes

- Mesure et traitement des données récoltées
- Paramétrage via le cloud

Plusieurs projet concret : Troyes, Grenoble, Brest

### Résultats

- Augmentation de la durée de vie du matériel
- Rentabilité en 3 et 5 ans
- 40% d'amélioration du traitement des pannes à grenoble

## Gestion de l'environnement

### Le produit

“Sonde syren”

Un capteur ultrason, robuste, étanche, Réseau IoT, envoie des infos de remplissage prises toutes les heures, envoyées toutes les 6 heures.

Doit tenir 10 ans sans aucune intervention humaine

Permet d'optimiser les tournées de remplissage

Application web sécurisée permettant de monitorer les données émis par les capteurs et des données plus abstraite : nombre de fois que le conteneur a été vidé, plein, transporté etc

### Résultat

- -20% de kilomètre parcourus
- -40% de temps de collecte gagné
- 1/3 des durée de vie

### Exemple concret

- 2400 conteneurs équipés de ces capteurs
- Gain de 30 à 60% dans la durée de vie du matériel

## Objectifs

Optimisation des tournées pour ne pas que le calcul dure des heures.

Question :



- Durée de vie de la batterie
- Concurrence
- Les données récoltées

## 06/01/2017: Kalray

[Lien vers les slides](#)

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCTSL0/document/TRANSPARENTS/2016-2017/MPPA\\_Getting\\_started\\_AC27.pdf](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMCTSL0/document/TRANSPARENTS/2016-2017/MPPA_Getting_started_AC27.pdf))

### MPPA Getting Start

Entreprise depuis 2008, 60 personnes, majorité dans la R&D, surtout ingénieurs

MPPA = 28 nm, conso faible -> pour embarqué

2 marchés :

- Data center très haut débit, traitement du flux réseaux
- Les très bonnes propriétés mémoires du chip, permet de le certifier pour des utilisations sécurisées

2 générations de proc:

- Andey : 1 ère version, pas assez bonne gestion des GPIOs
- Bostan :
  - 80 Gb ethernet
  - PCI
  - 600 MHz de fréquence
  - FPGA
  - DSP
  - 15 Watt

Plusieurs cartes pour différentes applications :

- Embarqué
- Turbo pour calcul intensif
- Une carte de développement
- ?

Pour utiliser ce matériel, il faut du logiciel:

- Driver

- Library optimisé en asm
- Débug
- Développement

## Architecture du MPPA

- DDR3 2 x 64-bit + ECC
- PCIe 2x8 lane
- 1/10/40 Go d'Ethernet
- GPIO
- NoC
- 16 cluster au centre
  - Chacun de ces 16 cluster a 16 coeurs
  - +1 Système Coeurs qui gère le NoC du cluster, et c'est le seul qui a accès à l'extérieur du coeurs
  - Chaque cluster partage 2 MB de mémoire avec un débit de 77 GB/s
    - Chaque coeurs sont des VLIW 32-bit
    - 5 instructions pas cycle à 600 MHz
    - 8 kB de cache instructoin
    - Instruction totalement propre à Kalray
  - Chaque cluster on 2 NoC : D(ata)-NoC avec DMA + C-NoC with control
- Autour des 16 clusters au centre, 2 cluster de 8 coeurs d'I/O
  - et chaque coeurs sont coupés en deux : donc  $2 * 2 * 4$  coeurs
  - Il y a donc 16 coeurs d'I/O
  - Sur un cluster : linux
  - Sur l'autre : Bare Metal
- Cloisonement dur des clusters centraux :
  - $4 + 4 + 4 + 4$  qui sont vraiment disjoint entre eux
  - Appréciable pour les utilisations critiques
- Le lien PCI permet de rendre le MPPA maitre ou esclave.

## Software

- Environnement C/C++
- Simulateur, profiling
- Driver
- Librairy d'optimisation
- OS : BareMetal, Linux ou intermédiaire

Outil de développement spécifique :

- Mode explicite : il faudrait faire des appels à des fonctions pour apporter les données dans le cluster.
  - Exemple : `frame = load_data(addr, size);`
  - C Low level : Programmation de type DSP
  - C Posix Level : Programmation de type CPU
- Mode implicite : On peut faire (grâce à la MMU) des accès “direct” à la mémoire.
  - Exemple : `frame = *image;`
  - OpenCL : Programmation de type GPU style
  - OpenDataPlane (ODP), Open API for networking
- Mode explicite :
  - Démarage des cluster et des I/O explicitement
- Mode inimplicite
  - Pas besoin de faire des appels explicites, la MMU s'occupe de tous les accès mémoire
- Outil pour compilation, de debug et de trace système
  - Les outils de traces permet de laisser des points de traces durant l'exécution pour une trait faible contre partie.

## Architecture mémoire du MMPA

Chaque coeur à un petit cache de niveau L1. Et tous ces caches ne sont pas cohérents. Toutes les coeurs d'un cluster partage aussi une même mémoire partagée

Attention, la gestion du cache est assez tricky, car les caches sont incohérents

Le DSM permet d'emuler dans le cache des coeurs d'I/O un cache L2 (pour cacher les pages de la DDR) et avoir toute la DDR adressage.

## Le Marché, voiture autonome

Capable de garantir en temps réel un débit avec une faible latence

Moins de 10 fois moins de consommations qu'un CPU 64-bit classique

## Le Marché, du stockage de data

La quantité de données qui arrivent pour être stocker, il faut traiter les données à la volée pour réduire la quantité de donnée stockée tout en lui donnant le maximum de valeur.

## 20/01/2017 : Thales

Ingénierie Logicielle en Développement Embarqué certifié critique

Laurent IMPERATRICE Charles DONNAT

## Thales

62 000 employés, 56 pays, R&D 707 millions 50% civil et 50% militaire  
Aéronautique, Espace, Transport, défense, ...

## Flight avionics

Proposer les différents équipements électroniques des avions.

Centrales inertielles, GPS, Capteurs, instruments de secours

C, C++, ADA 2012

## Un cadre normatif DO178B

Définir la criticité des pannes, par niveau et essayer de définir des mesures à prendre lors de la création du code pour que ça soit acceptable.

La DO donne des objectifs à tenir mais ne demande rien quand aux moyens d'y parvenir

### Processus de développement

Les objectifs sont : - Développer les exigences système, en créant un ou plusieurs niveaux d'exigences Software (dépendant du niveau de DAL) - Développer l'architecture Software - Produire le Code Source - Intégrer les composants Logiciel pour produire un Exécutable

Un artefacts logiciels = tous les produits du développement logiciel

Durant le processus de dev, plusieurs artefacts doivent être produit:

- Exigences de haut et bas niveau (HLR/LLR)
- Description de l'architecture logicielle
- Code Source
- Code Object Exécutable

### Processus de tracabilité

Il faut que chaque ligne de code soit justifiée.

Tout élément du binaire doit correspondre directement au code sources et doit correspondre à un besoin du produit, ni plus ni moins.

### Processus de vérification

On ne regarde pas le code, mais les exigences.

Le code doit être couvert à 100% par les tests.

Cette norme est imposé, acceptée par tous les industriels, et audités par des organismes externes.

Le cycle en V c'est bien mais :

- Trop chère à faire des changements à la fin
- Pas assez flexible

- Si le client n'est pas content à la fin, aie aie aie
- Gros problèmes entre le SW et le HW
- Démotivation du personnel

## Les méthodes propres à Thales pour parvenir à respecter ces normes

### Développement itératif incrémental

Le client veut quelques choses, l'équipe va le faire de façon pas à pas, morceau par morceau, mais il faudra les rassembler après : c'est l'itératif.

Le client veut quelques choses, l'équipe va commencer par une ébauche et va l'améliorer de plus en plus: c'est l'incrémental

Les deux ensemble font de l'itératif incrémental

### SCRUM

Une équipe, c'est :

- 1 product owner : personne qui connaît très bien les spéc du client
- 7 +/- 2 personnes, pluri-disciplinaires, co-localisées
- 1 ScrumMaster

Il peut y avoir des multi-Scrum:

- Plusieurs équipes
- 1 super ScrumMaster pour la synchronisation des équipes
- 1 super Product Owner garant du produit

Itération:

- Pas trop long ni trop court

Product backlog :

- Liste des fonctionnalités du produit
- Fait par le Product Owner
- Priorités et estimées
- Mise à jour à chaque itération

Stand up Meeting:

- Moment où l'équipe se réunit pour parler des problèmes rencontrés
- Faire un point sur : ce qui est fait, ce qui va, ce qui bloque, ce qui va être fait

## eXtreme Programming - Développement

### Problématique

- Développer le logiciel avec indépendance
- Transférer les compétences en continu

### Approche

- Pair programming
  - Sujet complexe
  - Formation
  - Partage de bonnes pratiques
- Polyvalence des développeurs
- Non Systématique

## eXtreme Programming - Test

### Problématique

- Plusieurs niveaux HLR/LLR
- Tester les exigences
- Assurer que les tests soient complets

### Approche

- Écrire en même temps les tests et les exigences
- Écrire les tests avant le code
- ...

## eXtreme Programming - Intégration continue

### Problématique

- Livrer un logiciel à tout moment

### Approche

- Intégrer de façon continue le code des autres

- Equipe mise à jour de façon continue
  - Feedback plus rapide
  - Correction rapide
  - Stop the line
- Production de warnings en continue

### **Difficultés**

- Durée des tests automatiques
- Stabilité de l'intégration continue, certain patch peut casser des choses qui passaient

### **Solutions**

- Optimisation des tests automatique
  - Parallélisation
  - Certain server font tourner des tests toutes la journée
  - Certain server font tourner des tests toutes la nuit
- Rejouer TOUS les tests avant la livraison

## **Lean - Value Stream Mapping**

### **Problématique**

Comment se mettre d'accord sur une liste d'activité à faire

### **Approche**

- Réalisation d'un VSM

## **Lean - Kankan (TAsk board)**

Un tableau qui représente chaque tâche à l'endroit où en est son avancement

## **Lean - Management visuel**

### **Problématique**

Comment rendre les problèmes visibles ?

### **Approche**

- Création d'un espace dédié par projet



## Lean - PDCA

### Problématique

Comment résoudre les problèmes montrés par toutes ces techniques

### Approche

- Plan : Décrire, Chiffrer la situation, Classer les causes racines
- Do : Liste des problèmes
- Check : Comment on peut vérifier si le problème se résoud ou s'empire
- Act : Comment réagir

## Conception et développement - Techno objet

Il faut absolument assurer que le code produit soit de qualité, testable, portable, évolutif, ré-utilisable.

Pour cela, Analyse, Conception, Implémentation

Programmation par contrat :

- Pré-condition
- Post-condition
- Invariant

## Des outils adaptés

- Pilotage : un projet est fini quand toutes les tâches sont faites, non pas quand on a passé toutes nos heures sur le projet. (EVM = Earned Value General)

□:

# Introduction

## EMPTY

␣:

# Introduction

## EMPTY

␣:

# Notes de Cours

Lien du cours sur [Github](https://github.com/moy/cours-tlm) (<https://github.com/moy/cours-tlm>)

## Cours 10/10/2016

Il faut créer/modéliser une hiérarchie des composants pour respecter le fonctionnement logique : video en même temps que audio et non séquentiel

## Les Communications

Pour les communications : - Un maître : initie la com - Un esclave : répond à un requête

Différent protocoles : - Bloquants : initie un échange et ATTEND la réponse - Non bloquant : ne bloque pas l'initiateur + canaux de com requête + réponse

## Les interruptions

Connections direct entre des composants par 1 fils

\* Permet de ne pas faire de polling \* Permet de faire des économie de l'énergie

Pas standardisé en TLM => on fait ce qu'on veut

### Question ?

Comment faire communiquer 2 procs ensemble ?

Il faut une mémoire et faire du polling si les proc sont initiateur seulement.

## Apport du TLM

- Développement du code embarqué en avance de phase
- Débuggage de l'intégration des composants
- Validation du RTL

## C++

Voir slide sur C++ du prof [sur son Github](#)

(<https://github.com/moy/cours-tlm/blob/master/02-c-plus-plus-handout.pdf>)

## Cours du 21/10/2016

### System C

Voir slide d'exemple de code [sur son Github](#)

(<https://github.com/moy/cours-tlm/blob/master/03-systemc-handout.pdf>)

#### Comment ça marche à l'intérieur ?

System C à un scheduler intern qui gère les processus : il a une “event list” et la liste des processus.

3 états dans le scheduler :

- Running - Sleeping - Éligible

Pour sortir de running c'est forcément que le processus à “rendu la main” en faisant un sleep.

Un proc qui est running ne peut que devenir Sleeping.

### Revision (encore) de C++

Voir les slides [sur le github \(page 0\)](#) du prof.

## Cours 16/12/16

### Emballage natif

Attention au boucle infinie dans laquelle le processus ne rend pas la main.

Dans ce cas, la simu ne s'arrêtera pas ...

Il faut faire appelle à `cpu_relax` pour rendre la main.

Inconvénients de l'emballage natif:

- Pas de support de l'asm
- Pas de visibilité des autres transactions (accès pile, tas, intructions (fetch)).
- Analyse de performances difficile

## Simulateur de jeu d'instructions

exemple : ISS Instruction Set Simulator

Émule de jeu d'instruction

Plusieurs niveaux de précision : - Instruction accurate - Cycle accurate - Cycle callable

Mais l'émulation a le principale inconvénients que c'est lent

## Autre solution :

- Just-in-Time (JIT) : Compiler le code, et découper le binaire en bloc de base et jongler entre l'ISS et les blocs de bases recompiler
- Ou autre translations dynamique : QMU etc

## Si il y a un OS ?

Avec ISS : ca marche mais trop lent

Solution native :

On pourrait porter Linux sur une architecture "System C TLM"

## Évaluation et exploration d'architecture

Comment est géré le temps en système C ?

wait(temps) -> change l'ordre des actions, vrai notion de temps

Rôle du temps en TLM ?

Plusieurs niveau de vue du temps :

- TLM Loosely Time (LT): temps moue, le temps à pas de signification, gros grain. (utilisé : programmation)
- Approximately Timed (AT) : Temps précis induits par la microarchi, communication à la taille du bus. (utilisé pour l'évaluation d'archi et de perf)

Problématiques :

- Exemple de lecture écriture en mémoire :  
En LT -> | READ | WRITE | En AT ->

|READ|WRITE|READ|WRITE|READ|WRITE|

en LT, tout pourrait être fait dans le même temps !

- Transducteur : C'est des convertisseurs qui permet de gérer le temps

## Bug or not Bug ?

Il peut y avoir des bugs HW et SW:

- HW : La simulation simule EXACTEMENT le HW qui avait un BUG, donc on peut corriger le HW -> cool
- HW : Erreur de programmation du simulateur
- SW : Bug de soft -> on debug
- SW : Bug du soft à cause du HW -> aie aie aie

Si le soft marche sur le TLM -> Il doit marcher sur le HW

Si SW marche PAS sur le HW -> Il ne doit PAS marcher sur le TLM

Si le SW marche sur le TLM mais pas sur le HW ... AIE AIE AIE

Donc un modèle TLM doit être fidèle ! Faire TOUT ce que le HW fait

Il peut avoir plus de comportement que le HW mais ne doit pas devenir iréaliste

Du soft erroné peut être simulé de 2 façons :

- ISS: Il y a un context switch toutes les instructions ou toutes les X instructions, dans ce cas, on peut "camoufler" des erreurs en simulation, alors que la vraie puce aurait plantée.
- en NATIF: Il faut faire des CPU\_RELAX() pour que le temps passe et pour que les autres processus puissent s'exécuter. Donc il peut également il avoir des camouflage d'erreur.

## Optimisation des performances

### Transaction bloc

Pour écrire des données à un composant sur le bus, on fait des socket.write(..) Or, si on fait un truc basique, pour chaque mot à envoyé, il faut faire une requête au bus, qui doit décoder l'adresse, etc etc ...

On pourrait faire un socket.block\_write(..) qui demande de faire un transfert depuis une adresse d'une quantité X. Ceci permet de faire 1 seul décodage d'adresse et donc de gagner pas mal de temps.

Cela rend donc l'écriture d'une zone mémoire atomique.

## Timing approximé

Context-switch est très cher

Donc on évite les wait.

Pour faire avancer le temps, on fait des wait(petit\_temps), à chaque tour de boucle.

**Solution : Quantum Keeping** On utilise un compteur sur 64 bits qui s'incrémente à la place du temps. Et de temps en temps on fait un wait de ce compteur qui à simuler le temps, et on remet le compteur à zéro. Cela fait faire un context-switch bien moins souvent.

## Parallélisation de SystemC

Les systèmes sur puces sont parallèles or SystemC n'a qu'un seul processus. Mais SystemC a qu'une notion de processus.

**Solution naive:** Chaque SC-THREAD crée des p\_thread -> bcp de p-thread donc ne passe pas à l'échelle

**Solution un peu mieux:** N processus = N processus SystemC

**Solution testé** On pourrait analyser le code et créer des processus qui exécute du code qui n'a pas de variables partagé. En théorie, bien. En pratique, impossible car bcp trop de dépendance entre les processus.

## Estimation de consommation d'énergie et de température

Modélisation de la plateforme en une machine à état à 3 états :

- Run : 3 Watt
- Idle : 1 Watt
- Wait : 0 Watt

Donc on fait l'intégral sur le temps passé dans chaque état pour connaître la conso sur un temps donné

Pour la température, on fait la somme de tout ce qui ajoute de la chaleur et de tout ce qui en enlève.



## Exam

- Questions de cours C++, SystemC/TLM, intervenant extérieur.
- Question temps simulé / temps wall clock
- Composant pour améliorer le TP3 : améliorer le memset
- SC\_MODULE, SC\_THREAD, SC\_METHOD, wait, notify, ...
- hal.h
- Principe de TLM2
- Confusion SW et HW

## Récapé TP

### TP1

Générateur -> Bus -> Memory

### TP2

Module LCD en plus, et la ROM

### TP3

- Intégration du logiciel embarqué
  - ISS
  - Native
- hal.h -> soft embarqué, doit être compilé avec le compilateur embarqué

## Intervenant ST:

Jerome CORNET

## Dernière nouvelle de l'industrie du semi-conducteur

### Conduite autonome

Ce qui se passe naturellement, pour pouvoir traiter toutes les données c'est de passer d'un CPU à GPU, puis Neural PE, traitement d'image.

Mais il y a aussi bcp de contraintes vis à vis de la quantité de traitement de données.

Sureté = Ne pas engendrer de dégâts en cas de défaillance.

Sécurité = Être protégé de menace extérieur.

Ce qui induit des chips gros, avec bcp d'IP très complexes, des moyens de communications nombreux et complexe, mémoire (importante).

### IoT

Beaucoup de cas d'utilisation possible : utilisateur privé, industrie, medical, ...

Ceci induit une très forte connectivité.

Ainsi que des problématiques de consommation

Induit l'augmentation du nombre de petits chip.

Des chips petits, sans mémoire, seulement quelques IP, ...

ISO 26262 c'est la norme de safty pour l'automobile

## Prototypage virtuel

Émulation = utilisation de logiciel qui existe déjà pour l'utiliser de la MEME façon qu'avec le HW d'origine.

Virtualisation = exécuter du soft mais dans le but de concevoir du SW non existant.

En virtualisation, on reproduit chaque petite tâche donc on peut trouver de vrais bug liés au temporel et à l'ordre.

Sert à faire du développement de logiciel embarqué en avance de phase.

Vu que c'est en avance de phase, ça permet de trouver des contradictions, des ambigüités du cahier des charges.

En post silicium, permet de comprendre ce qui se passe dans le chip.

Peut servir au client pour lui permettre de l'aider à développer le reste de son système.

Exploitation d'architecture = permet de ne pas s'intéresser aux données mais au débit et au transport de données.

### Métier de virtualisation à ST

Faire des modèles

Faire des couches au dessus de TLM (comme ensitlm) pour faire des choses réutilisable pour en faire des legos

## Grandes tendances

Fonctionnalités nouvelles:

- Simulation d'horloge : ex. Basse consommation
- Simulation de voltage
- Simulation des PIO Muxing (plusieurs fonctions sur un I/O)

Injection d'erreurs :

- Rien de plus simple, il suffit de changer quelques lignes dans le modèle.
- Simuler une intrusion dans le système. (changement dans la DDR)
- Tester des cas limites (cas des 16 collisions dans ethernet)

Intégration multi-système:

- Faire plusieurs modèles sur plusieurs simulateurs et les faire communiquer ensemble.
- Ça serait un vrai plus pour exploiter les many-cores.
- Permet de faire des simu pour des clients avec des morceaux de chez ST, et des morceaux chez les clients.

Dans l'automobile :

- Simuler les réseaux de nœuds d'une voiture qui comprend bcp de uC

## Conclusion

□:

# Introduction

## EMPTY

␣:

# Introduction

## Cours du 12/10/16

Récap sur séance précédent:

Quoi ?

Pourquoi ?

- Différence avec Système Embarqués/Centralisés et les Système distribués :
  - Communication entre processus
  - Mémoire partagée
  - Absence d'horloge partagée
  - Tolérance au faute Comment faire pour utiliser ces systèmes ?
- Spécification :
  - Communications ?
  - Synchronisation ?
  - Consensus ?
- Hypothèse

## Détection de fautes

- Détection de fautes parfait :
  - Specs:
    - Interfaces: notifie que le processus P a crashé
    - Propriétés:
      - Complétudes forte: Toutes pannes de processus sera détectée
      - Précision forte: Une panne machine est détectée comme panne processus
  - Hypothèse:
    - Panne franche sans aucune bornes

- Communication: fiables, borne sur le temps de propagation
- Temps de traitement: bornée et borne connue
- Algo

## Communication point-à-point fiable

- Spéc:
  - Interface:
    1. In : `send(m,p)` -> envoyer le message m vers le processus p
    2. out : `deliver(m,p)` -> permet de délivrer le message m émis par un processus p
  - Propriétés:
    1. Validité : si un processus correct émet un message m à destination d'un processus correct p alors ce dernier le délivrera.
    2. Intégrité : un processus ne peut délivrer un msg qu'au plus 1 fois et seulement si ce msg a été émis par un processus.
  - Hypothèse:
    1. Pannes franches
    2. Autres modèles de pannes
  - Algo + Implem + Preuve

TCP est un exemple de canal fiable donc interdit dans nos TP. On part d'UDP pour en faire un canal fiable.

### Ce qui est attendu pour le TP

1. Canal de communication fiable
  - Hypothèse
  - Algo
  - Implémentation : basé sur UDP
  - Évaluation des performances de l'implémentation
    - Débit (Netperf, ...)

- Ping = Latence

## 2. Détecteur de faute

- Hypothèse
- Algo
- Implémentation
- Validation expérimental

# Cours du 19/10/2016

## Diffusion de messages

### Best Exeffort Broadcast (BEB)

- Spécifications informelle:
  - N machines
  - Si un message est émis par une machine correcte, il doit être délivré (reçu) par toutes les machines correctes.
- Spécification formelle:
  - Interfaces:
    - *in*: broadcast(m): permet de diffuser le message m
    - *out*: deliver(m,p): permet de délivrer un message m émis par le processus p
  - Propriétés:
    - *Validité*: Si un message est émis par une machine correcte, il doit être délivré (reçu) par toutes les machines correctes.
    - *Intégrité*: Idem que le canal fiable.
  - Hypothèses:
    - Panne franche (pas de bornes sur le nombre de fautes)
    - Canaux fiables
  - Algo:

```

bf_broadcast(m){
    for(p=0; p<pmax; p++){
        cf_send(m,p);
    }
}
cf_deliver(m,p){
    be_deliver(m,p);
}

```

Rmq:

1. Tous les processus correcte ne délivreront pas forcément les mêmes message (ex: msg émis par machine fautive)
2. Un processus fautif peut délivrer des messages qui ne seront délivrés par aucun processus correct (ex: message émis par machine fautives)

### Reliable Broadcast (diffusion fiable)

Comment régler le point 1 précédent ? = Comment faire pour que tous les proc. correctes délivre les même msg ?

- Spéc = idem
- Propriétés:
  - *Accord*: Si un processus correct émet un message m alors il le délivrera
  - *Validité*: idem
  - *Intégrité*: idem
- Hypothèse :
  - Panne franches (pas de bornes)
  - Canaux fiables
- Algo :
  - Principe : Après avoir délivré le msg,le processus be\_broadcast le msg.
  - ATTENTION :
    1. Il faut s'arreter un jour
    2. Intégrité -> Il ne faut délivrer chaque message qu'une fois au plus

=> MEMOIRE INFINIE



## Cours du 24/10/2016

### Diffusion de message : 1 vers n

Pour garantir la propriété d'accord de RB, on doit multiplier le nombre d'envoi de messages par N, donc on divise par N la bande passante de notre canal.

De plus, pour savoir que l'on a déjà retransmis il faut mémoriser cette information. Plusieurs possibilités :

1. On enregistre tous les messages => problème de MÉMOIRE INFINIE
2. On numérote les messages par sources, et on les délivre par ordre croissant : coût mémoire =  $O(1)$  => pas vraiment efficace
3. On s'autorise une fenêtre de mémorisation => efficace

### BR plus efficace

Pour corriger le problème de bande passante de BR on doit trouver une solution. Une solution est de faire l'hypothèse que l'on a un détecteur de fautes parfait. Ainsi, si l'on avait reçu un message de la part d'un processus qui est tombé en panne, on le re-broadcast.

Attention, ici encore problème de MÉMOIRE INFINIE, pour se souvenir des messages que l'on devra renvoyer en cas de panne.

Du coup, ici aussi on doit trouver une astuce :

- Il faut vider la mémoire régulièrement
- Régulièrement, chaque processus informe les autres de façon "efficace" -> du type = j'ai reçu les 1000 msg de 3000 à 3999.
- **ATTENTION** que faire en cas d'absence de nouvelles d'un processus
- Pour cela, c'est le détecteur de faute parfait qui nous garantit que l'on va recevoir un message de tous les vivants.

Il reste encore un comportement non désirable :

- "Les processus fautifs peuvent délivrer des messages qui ne seront pas délivrés par les processus corrects
- Problème d'effets de bords
- Problème du redémarrage de machine

## URB = Uniform Reliable Broadcast

- Spéc = idem
- Propriétés:
  - *Validité*: idem RB
  - *Intégrité*: idem RB
  - *Accord Uniforme*: Si un processus (TOUS) délivre un message m, alors tous les processus CORRECT délivreront m
- L'idée pour cela est que la machine qui émet un message (qui peut tomber en panne après l'avoir délivré) doit attendre de recevoir tous les ACK pour le message m avant de le délivrer.
- Puis il notifie tout le monde que le message m est 'stable' = que tout le monde l'a ack
- Variante: les processus qui reçoivent m envoient un ack à tous les processus, et quand à leur tous ils reçoivent l'ack de m, ils le délivrent.

## Cours du 9/11/16

Tous ces algos ne respectent pas l'ordre d'envoi

Plusieurs solutions:

- FIFO : Tous les messages émis par un processus p sont délivrés dans l'ordre d'émission.
- CAUSAL : Pas dans ce cours, CAUSAL > FIFO
- TOTAL : tous les messages sont délivrés dans le même ordre pour tous les processus. Tous les BD exécutent les mêmes requêtes dans le même ordre. Mais pas forcément dans l'ordre FIFO.

## TOB = Total Order Broadcast

- Spéc = idem BEB, RB, URB
- Propriété : validité, intégrité, accord => idem RB ou URB
- - ORDRE TOTAL = si un proc p délivre m1 avant m2, alors aucun processus ne pourra délivrer m2 avant d'avoir délivré m1.
- Algo : Panne franche + Canal fiable + Détecteur de fautes + séquenceur.

- Un séquenceur est un processus élu par P.

Quand utiliser le détecteur de faute (P)?

On l'utilise quand il faut attendre l'ack des autres machines. Sinon on pourrait attendre un ack infiniment.

Mais supposer avoir un VRAI P, n'est pas tjs acceptable

## Paxos = algo de consensus

Consensus = TOB

(Paxos dans P) (avec P)

- Spéc : Interface :
  - Propose : permet de proposer une valeur
  - Décide : indique la valeur décidée
- Propriétés :
  - Toutes les machines correctes décident la même valeur.
  - La valeur décidée a été proposée par une machine

Paxos = implante la spec du consensus en faisant les hypothèses suivantes:

- Canale fiable
- Pannes : "crash recovery"

Paxos à besoin de  $2f+1$  machine pour tolérer  $f$  fautes

=> Suppose  $f=1$  : -> Paxos a besoin de 3 machines

- Si 0 panne : consensus faisable
- Si 1 pannes : consensus faisable
- Si  $>1$  pannes : blocage MAIS PAS D'ERREUR ! BLOCAGE SEULEMENT

Paxos se décompose en 4 règles :

Pour  $f=1$ ,  $Av$  = Accepted value,  $Sn$  = Séquence number

1. Si  $Sn$  PREPARE  $>$   $Sn$  stocké en local -> OK +  $Av$  sinon KO
2. Si  $f+1$  OK alors le proposeur peut émettre (ACCEPT,  $Sn$ ,  $v$ )  
avec  $Sn$  = le num utilisé dans le proposeur value  
 $v$  = si  $f+1$  (-1) alors  $v$  = ce que veut proposer le proposeur  
sinon la valeur associé avec le plus grand  $Sn$  reçu
3. Si  $Sn$  du ACCEPT  $\geq$   $Sn$  stocké alors  $Av(Sn, v$  du accept) et OK (sinon KO)

(lien de la video)[<https://www.youtube.com/watch?v=cj9DCYac3dw>]

## Cours du 16/11/16

### Notion de performance

Exemple : Protocole de performance basique

Métrique :

- Débit : nombre de message déivré par unité de temps => systèm e doit être saturé
- Latence : temps entre l'émission et la délivrance par le "dernier" proc.  
=> le système ne doit pas être saturé
- Temps de réponse :

Système physique sur lequel s'exerce le protocole :

Cas d'étude : Datacenter : simple et réaliste

Modèle de rondes : chaque processus peut :

- émettre 1 message AU PLUS au début de chaque ronde.
- recevoir 1 message AU PLUS à la fin de chaque ronde.

Pourquoi modéliser :

1. Avoir une approximation des performances sans implémenter le système
2. Résonner sur latence vs. débit : débit optimal, latence optimal ...

Pour optimiser la latence :

p1 envoie m à p2 au ronde 1

p1 envoie m à p3 et p2 envoie m à p4 au ronde 2

Latence = 2, Debit = 0.5 m / ronde

Pour un débit optimal :

p1 envoie m à p2

p2 envoie m à p3 et p1 envoie m2 à p2

p3 envoie m à p4 et p2 envoie m2 à p2 et p1 envoie m3 à p2

p3 envoie m2 à p4 et p2 envoie m3 à p3

p3 envoie m3 à p4

Latence = 3 rondes, Débit = 1 msg / ronde

Le débit maximal entre N machine est  $= N/(N-1)$

⌈:

# Introduction

## Ce qui est attendu pour le TP

1. Canal de communication fiable
  - Hypothèse
  - Algo
  - Implémentation : basé sur UDP
  - Évaluation des performances de l'implémentation
    - Débit (Netperf, ...), débit en fonction de la taille des messages
    - Ping = Latence
2. Détecteur de faute
  - Hypothèse
  - Algo
  - Implémentation
  - Validation expérimental
3. Petit rapport
4. Archive du code avec README

☐:

# Note de Cours

## Cours du 10/10/16

### Nécessité de “dépacker” la puce

- Carte à puce
  - Sous les 6 plots d’une carte à puce se trouve la puce
  - Une fois que le circuit (puce) est sortie, il y a que 6 broches moins 2 (VCC, GND), ce qui fait 4.
  - Peu compliqué
- Carte RFID
  - Traitement chimique pour supprimer le plastique et récupérer la puce RFID

Une fois que le circuit est “nu” on peut l’exploiter pour les attaques.

**⚠ Important:** Le fait d’enlever le package, le circuit sera exposé à la lumière. Les concepteurs ont pu mettre des capteurs de lumière ou d’UV pour que le circuit se bloque en cas d’exposition à la lumière

### Observation du circuit

Avant, on pouvait observer directement le circuit et comprendre son fonctionnement. Or mtn, le circuit est fait de bcp de couches qui sont homogénéisés pour ne plus pouvoir récupérer d’information par observation. Le métal ajouté s’appelle les : “Dummies”

### Enlèvement des couches

Par des traitements chimiques, on peut enlever les couches de métal, d’isolant etc ... Il est ensuite possible d’observer avec des microscopes assez puissant les différents éléments qui composent le circuit Par cette méthode on peut :

- Repérer l’emplacement des composants
- Récupérer des données stockés en dure dans des ROM

Pour rendre la tâche plus difficile à l'attaquant, on peut stocker les octets en ROM dans des ordres non convencionel : Trumbling Mais si on faire la même observation sur le décodeur de ROM, on peut encore une fois retrouver les données de la ROM

## Autres interventions physiques

- Couper des interconnexions internes
- Ajouter ou modifier des interconnexions internes Pas toujours facile sans abimer les autres zones du circuit Mais l'une des utilisations pourrait être de détruire ou reconstruire des fusibles de sécurité.

## Observation dynamique du circuit

Avec des SEM, Microscopes électroniques, on peut observer les niveaux logique dans un circuit au runtime. Très utilisé pour aller lire des mémoires programmables qui pourraient contenir des clés secrètes.

## Attaque par canaux caché/auxiliaire

- Mesure du temps d'exécution
- Mesure de consommation de courant (SPA, DPA, CPA)
- Mesure des émissions EM
- Mesure de son, chaleurs, photons
- Ou d'autres encore ... ?

## Quelle conditions pour faire de telles attaques ?

- Il dois y avoir un lien entre la clé secrète et une grandeur physique
- Et pour savoir quelle grandeur doit être mesuré, il faut connaitre quel algo est utilisé

## Type d'attaque par canaux auxiliaire

- Simple
  - Grandeur directement mesurable
- Différentiel



- Nécessite plusieurs mesures, qui peut être très important

## Les attaques par canaux auxiliaire

### Attaque par analyse de temps

- Dans des boucles If Then Else il peut y avoir des temps d'exécution différents en fonction des valeurs d'entrée
- Il faut des fois accepter de perdre en performance pour plus de sécurité :
  - Implém 1: if(x=1) then a=a+b;
  - Implém 2: t[0]=a ; t[1]=a+b ; a=t[x] ; Implémentation 2 est sécurisé vis à vis de la mesure de temps.  
En contre partie, plus d'accès mémoire
- Autre exemple : code PIN de la carte bleu
- Même si la mesure ne permet pas de dévoiler la clé, ces méthodes de mesure de temps peut permettre de construire des statistiques qui permettront de réduire considérablement le nombre de possibilité à essayer par brute-force

### Attaque par analyse de consommation

- SPA : Simple Power Analysis
  - Un transistor consomme du courant lorsqu'il commute
  - En mesurant le courant, on peut remonter aux "bits" du système
  - On repère un motif qui révèle la clé, vu qu'on connaît les algo utilisés
- DPA : Differential Power Analysis
  - Il faut des modèles pour lier la physique aux informations qui se propagent dans le système
  - Il faut ensuite trouver l'opération qui nous permettra de trouver l'information voulu
  - Par exemple  $M' = f(M \text{ XOR } K)$
  - On découpe ce genre d'opération en sous bloc, on fait des hypothèses de clés et voit l'influence sur la consommation de courant

### Attaque par analyse d'émission électromagnétique

Méthode plus complexe car dépend beaucoup du matériel et de son utilisation

Mais les courbes obtenues sont bien plus précises car aucun "filtre".

En effet, lors de la mesure de courant, les capacités du système filtrent le courant mesuré.

Ce n'est pas le cas des mesures EM.

### Attaque par analyse d'émission lumineuse

Permet de visualiser l'état de certaines informations transitant dans le circuit.

## Attaque par perturbation du circuit

- Modification des conditions de fonctionnement
  - Température
  - Tension d'alimentation
  - Fréquence d'horloge
  - ...Ces attaques peuvent être protégées par utilisation de capteurs
- Sans modification des conditions et en injectant des erreurs dans les systèmes
  - Glitches (power and clock)
  - Injection de particules : UV, etc Ici aussi il peut y avoir des capteurs pour protéger

### Pourquoi injecter des fautes ?

Exemple de l'attaque de Bellecore sur RSA CRT

Avec un flash on perturbe un chiffrement et avec quelques propriétés mathématiques, on peut retrouver la clé très facilement.

### Une attaque idéale serait

- Localisée (x,y)
- Localisé en temps (début et durée)
- Type de la faute : stuck at, injection d'erreurs
- Bas coût
- Rapide

## Cours du 17/10/16

### Création d'erreur

SET: Single-Event Transients

SEU: Single-Event Upset

MBU:

Attaque software:

- SW flaw exploitation, test protocols
- Buffer overflow
- Trojan horses (e.g., hidden in games,...)
- Virus, code agressif, attaque par réseaux

#### Trojans (cheveau de troi)

- HW spys / wireless reporting
- Hidden “bombs” (passage à l’an 2000)
- PUF = Physically Unclonable Functions:
  - Pucés qui permet d’identifier un objet
  - Il faut faire un circuit, et le remplir aléatoirement, qu’on ne contrôle pas
  - Un hacker ne pourra que reproduire le circuit et pas l’aléatoire
  - Ce nombre doit être reproductible mais unique (différent des autres produit)
  - Ex: Initialisation d’un plan mémoire, ...

### Problématiques des tests

Il y a un vrai problème avec le test de circuits critiques, les moyens de test pourrais être une énorme opportunité pour un attaquant.

Par exemple, la chaîne de scan peut faire sortir des clés de chiffrement

Il faut donc mettre en place des mesures de test sans que ça influe la sécurité

## Pourquoi étudier les modèle de fautes

Radiations de particules qui peuvent faire des fautes par exemple

Le choix du modèle est important : Bit-flip, Bit-set, Bit-reset

Mais en pratique, c'est pas tjs facile de faire certaine attaque.

## Cours du 07/11/16

Contre-mesure

## Analyse pas canaux auxiliares

Attaque liées à certaines grandeur physique

- Time
- Power
- EM
- ...

Le principe de ces attaques est de trouver un lien entre ces grandeurs physique et les données sensibles.

Les contres mesures consiste donc à rendre le lien bcp plus difficile à trouver.

SPA = Simple Power Analysis -> pas très utile pour les algo symétrique

DPA = Differential Power Analysis

Il faut donc rendre complexe la mesure de ces grandeurs physiques

Sécurisation de 'scan chain' qui donne accès au circuit

## Structure de tests

Scrambling méthode :

Faire des scan chaine non linéaire, mais aléatoire par segment

Scan-Enable Tree:

Vérifier l'intégrité du signal

Spy Flip-Flop:

tester si le contrôleur de test à un comportement "attendu", "correcte"

Built in Self Test (BIST):

Architecture en 3 'blocs' :

- TPG: Génère des vecteurs de tests
- DUT: Devise Under Test
- ORACLE: Comparateur de sortie

## Attack par faute

### AES c'est quoi ?

Algo symétrique = même clé pour chiffrer et déchiffrer

Text = 128b

Clé = 128/192/256b Tour pour un chiffrement = 10/12/14 rounds

1 round =

- SubBytes: substitution non linéaire de byte
- ShiftRows: rotation des lignes
- MixColumns: multiplication linéaire par colonne
- AddKey : ajout de la clé modifié

## Code correction/détection d'erreurs

EDC

## Cours 15/11/2016

lien des slides

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMSSE/document/Transparents\\_SSE\\_16-17.pdf?cidReq=ENSIMAG5MMSSE&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMSSE/document/Transparents_SSE_16-17.pdf?cidReq=ENSIMAG5MMSSE&id_session=0&gidReq=0&origin=))

## Cours 07/12/2016

Cours/TD

### Exemple 1

Full Adder

TMR : Triple Modular Redudency

On triple tout, on ajoute un voteur et on met ca dans une boite Attention, à la synthèse il va y avoir une optimisation de tout se qui est redondant. Donc il faut ajouter des contraintes à l'outil de synthèse.

Un moyen de vérifier peut être de comparer la surface du circuit avec et sans redondance.

On peut aussi comparer le chemin critique. Celui du TMR devrait être un petit peu plus grand à cause du voteur.

Dans un flow FPGA il peut y avoir aussi des contraintes de mappings. Si deux fonctions redondante sont mappés dans la même LUT, dans le même CLB, la redondance perd tout son sens.

Il peut aussi y avoir un deuxième problème, si au routage on a les fonctions redondantes sur des LUT qui utilisent les mêmes "switch box" entre les LUT, la redondance perd son sens.

Il faut donc que chaque fonction de redondance soit dans les LUT différentes et indépendantes.

□:

# Introduction

## EMPTY

␣:

# Introduction

## EMPTY

␣:



## Overview

This site provides documentation, training, and other notes for the Jekyll Documentation theme. There's a lot of information about how to do a variety of things here, and it's not all unique to this theme. But by and large, understanding how to do things in Jekyll depends on how your theme is coded. As a result, these additional details are provided.

The instructions here are geared towards technical writers working on documentation. You may have a team of one or more technical writers working on documentation for multiple projects. You can use this same theme to author all of your documentation for each of your products. The theme is built to accommodate documentation for multiple products on the same site.

## Survey of features

Some of the more prominent features of this theme include the following:

- Bootstrap framework
- [Navgoco multi-level sidebar](http://www.komposta.net/article/navgoco) (<http://www.komposta.net/article/navgoco>) for table of contents
- Ability to specify different sidebars for different products
- Top navigation bar with drop-down menus
- Notes, tips, and warning information notes
- Tags for alternative navigation
- Advanced landing page layouts from the [Modern Business theme](http://startbootstrap.com/template-overviews/modern-business/) (<http://startbootstrap.com/template-overviews/modern-business/>).

## Getting started

To get started, see [Getting Started \(page 3\)](#).

□:

# Introduction

## Voir cours papier

☐:

# Notes de cours

## Cours du 12/10/16

### Lien du cours

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/12.10.16.pdf?cidReq=ENSIMAG5MMVSE&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/12.10.16.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=))

Un bug ça coûte cher, mais peut être moins cher que de la vérif.

Exemple : - Avion = Énormes vérif et contrôle et norme - Voiture = Pas de normes ... mais ça commence un peu - IoT = Pas de vérif car temps de vie "trop court"

Difficulté de l'embarqué et de devoir tester avant et en dehors de son contexte normal d'exécution.

Comment savoir si le système fonctionne comme on a prévu qu'il fonctionne.

Mais il est aussi difficile de savoir ce que l'on aimerait que le système fasse.

La vérification c'est une comparaison entre : - La description entre ce que je veux - Et l'implémentation du système

Si non égale, se poser des questions, si oui c'est pas forcément génial !

Prenons un exemple : \* Système de stabilisation d'un avion -> les mouvements de l'avion doivent être pris en compte pour que l'objectif soit atteint de façon sûre. \* ABS d'une voiture -> la voiture doit réagir correctement en sécurité \* Téléphone mobile faisant de la vidéo : Correct = ? -> faut juste que ce que l'on voit soit "beau" \* Carte électronique -> il faut connaître les attaques pour pouvoir se défendre.

### Statique ou dynamique ?

Statique = Regarder le programme sans le faire tourner Dynamique = tout le reste : test à l'exécution, debug etc ...

Est-ce que l'on peut savoir si il y a une division par zéro ? \* Peut pas être trouvé statiquement !

Mon programme a-t-il besoin d'une mémoire finie ?

\* Peut pas trop être détecté en statique, sauf si pas d'allocation mémoire, ni de récursion Mon programme aura terminé en moins de 3 secondes ?

\* Indécidable statiquement

Le type de réponse d'un test de validation serait du type :

\* Il existe un morceau de code qui plante \* Il pourrait y avoir un morceau de code qui fait planter \* Pour TOUTES les entrées possibles, mon système est correcte.

Pour savoir si une erreur est accessible par un programme il faut :

- Graph du programme
- Graph d'une condition à respecter
- Faire le produit synchrone des deux
- Regarder plus finement les liens menant à un état d'erreur pour savoir les transitions sont possibles, et donc si l'état d'erreur est accessible

## Cours du 19/10/2016

lien du cours (<http://www-verimag.imag.fr/~raymond/edu/sleverif/slides-verif-sle-po.pdf>)

### Introduction

3 notions de corrections d'erreurs:

- Propriété générique: débordements arithmétiques, famine, bocage, etc ...
- Propriété spécifique: Overflow, etc ...
- Propriété runtime: manque de mémoire, etc ...

On peut essayer de raisonner par fonctionnalité : Modèle Fonctionnel

- S'affranchir du système concret, pour juste raisonner sur les fonctions du système
- L'exécution d'un système réactif est essentiellement une séquence de réaction entrées/sorties:
- Si indéterministe : relation e/s - Si déterministe : relation  $e \rightarrow s$

Plan du cours:

- Rappel sur Lustre
- Exprimer/Vérifier des propriétés dynamiques: les quelles ? limitations ?
- Abstraction finie
- Technique d'exploration des systèmes finis
- Des exemples, exercices, etc ...

## Language Lustre

### Pourquoi Lustre ?

Simple, proche des maths, vision par flow de données

Types et opérateurs:

- Bool, int, real
- and, or, not, +, -, \*, /, if-then-else
- pre
- '->'

Hystérésis:

Pour ne pas osciller on utilise un système d'hystérésis = seuils de tolérance décalés

Pour passer de l'état 1 à l'état 2 la condition ne sera pas la même que pour passer de 2 à 1.

## Définir les Propriété fonctionnelle

Propriété fonctionnelle = ensemble de comportements correct

Vérifier un propriété fonctionnelle = Vérifier que l'ensemble des comportements possibles est inclus dans l'ensemble des comportent corrects

Propriété d'invariance(sureté):

- Propriété d'état:

- Une configuration (ou état) d'un système est une valuation particulière des entrées, sorties et mémoires internes du système.

- Une propriété d'état est une relation (i.e. un ensemble) de configurations.

- Propriété de sureté:

- Une propriété de sûreté exprime le fait qu'une propriété d'état est invariante au cours du temps.

- Ou, de manière équivalente, qu'une configuration (redoutée) n'arrive jamais

- n.b. le mot anglais est « safety »

## Modèle équationnel

Système réactif est caractérisé par  $S$  = Ensembles des entrées,  $S$  = Ensembles de ces sorties,  $M$  = Ensemble des mémoires internes. Si ce système est déterministe alors il peut être modélisé par  $(E, S, M, Mo, f, g)$ .

$Mo$  = Mémoire initiale

$St = f(Et, Mt)$  est la fonction de sortie

$Mt+1 = g(Et, Mt)$  est la fonction de transition

Grâce à Lustre, c'est assez simple.

## Cours du 16/11/2016

## Interprétation abstraite

LIRE LE PAPIER [ICI](#)

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/Cache-absint.pdf?cidReq=ENSIMAG5MMVSE&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/Cache-absint.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=))

[lien vers les slides de tout le cours](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/12.10.16.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=0)

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/12.10.16.pdf?cidReq=ENSIMAG5MMVSE&id\\_session=0&gidReq=0&origin=0](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/12.10.16.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=0))

[lien vers les polyèdres](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/nicolas-polyedres-a.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=0)

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/nicolas-polyedres-a.pdf?cidReq=ENSIMAG5MMVSE&id\\_session=0&gidReq=0&origin=0](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/nicolas-polyedres-a.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=0))

Pour un état donné, on ne peut pas calculer l'ensemble des valeurs possibles  
Donc, on calcule un sur-ensemble :

- signes : Numérique
- intervalles : Numérique
- polyèdres convexes : Numérique
- truc inventés : WCET par exemple

Si jamais ce sur-ensemble est nul, alors l'ensemble, qui est incluse dans le sur-ensemble, est vide lui aussi, donc cet état n'est pas atteignable.

Pour ranger les ensembles, on utilise un treillis complet.

Théorème 1: Si on a un treillis complet,  
si on a une fonction monotone, alors il existe un point fixe max et min.

Théorème 2: Pour trouver le point fixe min et max, on calcul la limite de  $f(\text{bottom})$   
et  $f(\text{top})$

## Cours du 23/11/2016

### Pour faire une interprétation abstraite de signes

On fait le graph

On met des étiquettes sur chaque noeud

On fait un "photo" à l'instant  $t$  c'est l'ensemble des états et des étiquettes d'état.

Puis la photo à l'itération suivante.

Si ça ne change plus => point fixe.

On peut alors remplir un tableau avec tous ces photos.

### How to compute WCET

Souvent les programmes on la tête de :

```

init();

while(1){
    input();
    compute();
    output();
}

```

Le WCET peut être déterminé de façon “grossière”, avec bcp de marge, par l’interprétation abstraite

```

for (int i=0; i<100; i++) {
    if (i%2==0){
        s();
        // Grand cout
    } else {
        p();
        // Petit cout
    }
}

```

Un truc bourrin dirait :  $WCET = 100 * \max(s,p) = 100 * p$ .

Un truc malin dirait :  $WCET = 50 * (p + s)$ , et c’est bcp mieux !

## Analyse d’un programme par interprétation abstraite

cf. [ce lien](#)

([http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/Cache-absint.pdf?cidReq=ENSIMAG5MMVSE&id\\_session=0&gidReq=0&origin=](http://chamilo2.grenet.fr/inp/courses/ENSIMAG5MMVSE/document/TRANSPARENTS/Cache-absint.pdf?cidReq=ENSIMAG5MMVSE&id_session=0&gidReq=0&origin=))

Voir le programme comme un graph où sur chaque transition il y a 1 seul accès mem

On ajout les étiquettes tel que : 1 étiquette regroupe l’ensemble des variables qui sont SUPRÊMENT dans le cache.

Cache Fully associative, LRU. Mémoire de taille m.

Cache de taille n. (n lignes)

Le plus vieux éléments du cache est en bas.

Donc en cas de hit, on le remonte tout en haut et descend tous les autres de 1

En cas de miss, on rentre le nouveau tout en haut, et le plus vieux dégage.

Question: Combien d'état concret du cache (dans les conditions du papier):  $m+1 * m * m-1 * \dots * m-n+1$

C'est donc un ensemble fini, mais très gros.

Question: Peut on faire une analyse avec les vrais états du cache.

Question: A quoi ressemble l'état du cache, d'après ce graphe.

Soit on accède à X soit à Y.

E1 est un cache vide.

Quel tête à E2 ? Pour représenter cela, il faut avoir un “cache abstrait”

Il permet de dire : à la ligne 1 du cache, j'ai soit X, soit Y.

Cache abstrait = cache à n lignes ou dans chaque ligne on peut mettre par exemple:

“x | y”.

Union abstraite = Join =  $J(E1 *, E2 *) = \text{MUST analysis}$

On fait l'intersections de  $E1 *$  et de  $E2 *$  sur chaque ligne et on le met sur la case la plus vieille possible (Pire cas).

Exercice : `c while(e) { b; c; a; d; c; }` avec 4 lignes de caches : `[| | | |]`

Avant le while le cache est [ b,d c,z]



```

      . [llb,dlc,z] ; J([llb,dlc,z],[cldlalb]) =
[lldlb]
      ^      \
      /      \ e
      |      . [elllb,d,z] ; [ellld]
      |      |
      |      | b
      |      . [blelld,z] ; [blell]
      |      |
      |      | c
      |      . [clblel] ; [clblel]
      |      |
      |      | a
      |      . [alclble] ; [alclble] IDEM boucle 1
      |      |
      |      | d
      |      . [dlalclb] ; ...
      |      |
      |      | c
      -----

```

Des la deuxième itération, on a trouvé le point fixe.

Noter également que si l'on a 1 lettre par ligne, c'est que l'on a une représentation concrète

MUST analysis = On garde tout ce que l'on est SUR d'avoir

MAY analysis = On garde tout ce que l'on pense pouvoir avoir

[]:

# Introduction

## EMPTY

␣: