

ENSEIRB-MATMECA
IT 219

**Logiciels Libres
pour l'embarqué**

Patrice KADIONIK

email : kadionik@enseirb.fr
http : <http://www.enseirb.fr/~kadionik>

HISTORIQUE

- V.5.0 01/11 : Création depuis cours SE. 4

CHAPITRE 0 : INTRODUCTION

INTRODUCTION

- Cette formation a pour but de présenter tous les éléments techniques pour appréhender le monde des systèmes embarqués :
 - Les systèmes embarqués aujourd'hui : systèmes embarqués, Temps Réel, Linux embarqué, Codesign...
 - Le codesign : le mariage du matériel avec le logiciel.
 - La mise au point des systèmes embarqués : conception, les outils de debug, trucs et astuces.
 - Linux embarqué : Les concepts. Le panorama. Présentation de la mise en œuvre de μ Clinux comme exemple.
 - Le Temps Réel et Linux. Les concepts. Le panorama. Présentation de la mise en œuvre de RTLinux comme exemple.

CHAPITRE 1 : LES SYSTEMES EMBARQUES AUJOURD 'HUI

IMPORTANCE DU MARCHÉ DE L'EMBARQUÉ

- Les systèmes (numériques) embarqués ont vu leur importance progresser au rythme de l'importance prise par les microprocesseurs.
 - 1971 : premier microprocesseur 4 bits 4004 d'Intel à 92,5 kHz vendu 200 \$. Le succès a été là tout de suite.
 - Juin 1978 : premier processeur x86 8086 à 4,77 MHz (technologie 3 μ m, 29000 transistors), bus d'adresse 20 bits à 9,1 Mo/s, bus de données 16 bits.
 - Juin 1979 : 8088 intégré dans le premier IBM-PC en 1981.
 - Motorola, Zilog, TI ont emboîté le pas...
- Le marché des microprocesseurs est un marché qui croît de façon exponentielle.

IMPORTANCE DU MARCHE DE L'EMBARQUE

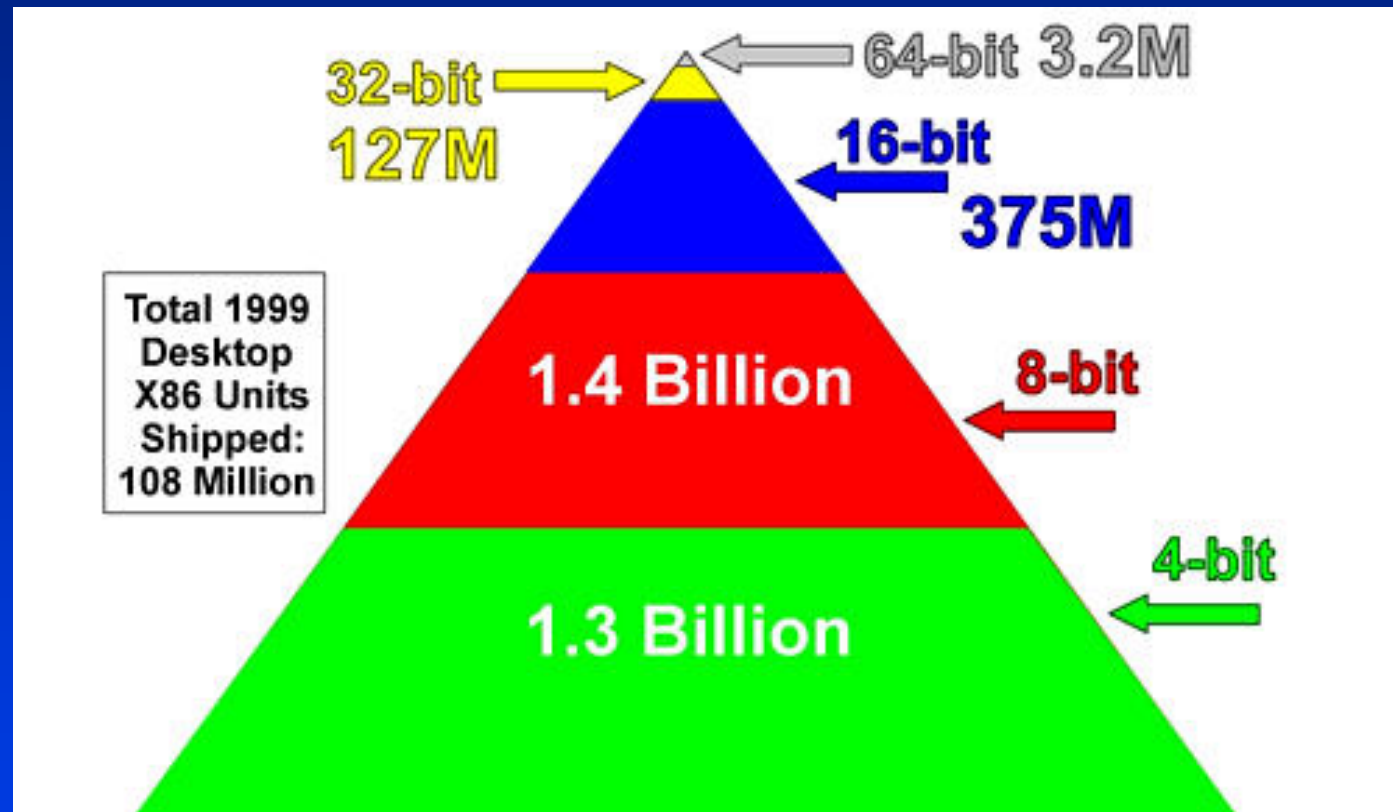
- Deux lois empiriques sont vérifiées depuis 30 ans (en plus de la loi de Moore) :
 - Loi de JOY : la puissance CPU en MIPS double tous les 2 ans.
 - Loi de RUGE : on a besoin d'une Bande Passante de 0,3 à 1 Mb/s par MIPS.
- Le marché du microprocesseur a aussi tiré le marché des systèmes embarqués (et des télécommunications !).

IMPORTANCE DU MARCHÉ DE L'EMBARQUÉ

- Grâce aux progrès de l'intégration sur silicium, on est passé rapidement du processeur 4 bits au :
 - processeur 8 bits.
 - processeur 16 bits.
 - processeur 32 bits.
 - processeurs 64 bits.
- Il ne faut pas croire que le marché du microprocesseur se résume à celui du PC via les processeurs x86.

IMPORTANCE DU MARCHÉ DE L'EMBARQUÉ

- La figure suivante démontre le contraire (année 1999) :



IMPORTANCE DU MARCHÉ DE L'EMBARQUÉ

- Il a été vendu 108 millions de processeurs x86 pour le marché du PC contre 1,4 milliard de processeurs 8 bits pour le marché des systèmes embarqués (appelé aussi marché de l'embarqué) !
- On voit ainsi que 2 % des processeurs vendus sont pour le marché du PC. Dans 85 % des cas, Microsoft Windows est utilisé.
- Pour 98 % des autres processeurs vendus, on utilisera généralement un autre système d'exploitation (OS : *Operating System*).
- On trouvera ici dans 60 % des cas un OS propriétaire. Beaucoup optent pour des OS libres comme Linux pour limiter les coûts...

IMPORTANCE DU MARCHÉ DE L'EMBARQUÉ

- Pour 2004, il a été vendu environ 260 millions de processeurs pour le marché du PC grand public à comparer aux 14 milliards de processeurs tout type confondu (microprocesseur, microcontrôleur, DSP) pour le marché de l'embarqué.
- Le marché des processeurs pour l'embarqué selon *Electronics.ca Research Network* devrait croître de 6 % en 2005 pour un chiffre d'affaire mondial de 18 milliards USD !

IMPORTANCE DU MARCHE DE L'EMBARQUE

- Moins de 10 % des processeurs vendus sont des processeurs 32 bits pour près de 31 % du chiffre d'affaire sur les processeurs.
- Cette part du chiffre d'affaire est estimée à près de 48 % pour 2008 : cela montre la migration rapide vers ces processeurs 32 bits dans l'embarqué, condition nécessaire pour pouvoir mettre en oeuvre Linux.
- Si l'on regarde le prix moyen d'un processeur tout type confondu, on arrive à 6 USD par unité à comparer au prix moyen de 300 USD par unité pour un processeur pour PC. Le marché du processeur pour PC est très faible en volume mais extrêmement lucratif !

LE CHOIX D 'UN PROCESSEUR POUR L 'EMBARQUE

Embedded Processor	System Requirement	Feature	Benefit
Microcontroller	I/O Control	I/O Ports with bit-level control	Efficient control of external devices Direct interface to actuators, switches and digital status signals
	Peripheral Communication	Serial Ports : SPI, I ² C, Microwire, UART, CAN	Hardware support for expansion & external device networking and communications
	Precision control of motors and actuators	Sophisticated timers and PWM peripherals	Low software overhead
	Quickly resolve complex software program control flow	Conditional jumps Bit test instructions Interrupt priority control	Efficiently implement control oriented algorithms
	Fast response to external events	External interrupts with multiple priority levels	Program control immediately redirected on event occurrence with minimal overhead
	Conversion of sensor data	Analog-to-Digital (A/D) Converters	Hardware support for external sensors

LE CHOIX D 'UN PROCESSEUR POUR L 'EMBARQUE

Embedded Processor	System Requirement	Feature	Benefit
DSP	Software Filters	Multiply/Accumulate Unit Zero-overhead loops	Digital filtering in few cycles
	Interface to codecs	High-speed serial ports	Hardware support for translation of analog signals
	High data Throughput from serial ports	Peripheral DMA	Less wasted cycles fetching data from serial ports
	Fast data access	Harvard architectures and variants	Fast execution of signal processing algorithms

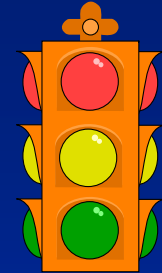
LE CHOIX D 'UN PROCESSEUR 32 BITS POUR L 'EMBARQUE

Besoin	Miniature	Petit	Moyen	Haut de gamme	PC embarqué	Embarqué haute disponibilité
Taille RAM	<0,1 Mo	0,1-4 Mo	2-8 Mo	8-32 Mo	16-64 Mo	> x Mo
Taille ROM/FLASH	0,1-0,5 Mo	0,5-2 Mo	2-4 Mo FLASH	4-16 Mo FLASH	xx Mo	Go-To
Processeurs	DragonBall 68K Mcore ColdFire ARM		MIPS Hitachi SH x86 PowerPC		Pentium PowerPC	
Caractéristiques matérielles	MMU optionnelle		Ardoise Internet Carte unité centrale System on Chip (SoC)		CompactPCI	
Exemples d'applications	Caméra numérique PDA Téléphone		Routeur Décodeur Stockage en réseau Imprimante en réseau		Commutateur téléphonique Routeur haute performance Serveur central	

SYSTEME EMBARQUE : DEFINITION

- Un système embarqué peut être défini comme un système électronique et informatique autonome ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur (PC).
- Le système matériel et l'application sont intimement liés et noyés dans le matériel et ne sont pas aussi facilement discernables comme dans un environnement de travail classique de type PC.
- *Un système embarqué est donc un système électronique et informatique autonome, qui est dédié à une tâche bien précise.*

SYSTEME EMBARQUE : DEFINITION



- Un système embarqué :
 - Est un système numérique.
 - Utilise généralement un processeur.
 - Exécute un logiciel dédié pour réaliser une fonctionnalité précise.
 - Remplace souvent des composants électromécaniques.
 - N 'a pas réellement de clavier standard (BP, clavier matriciel...).
 - L 'affichage est limité (écran LCD...) ou n 'existe pas du tout.
 - N 'est pas un PC traditionnel.

SYSTEME EMBARQUE : DEFINITION

- Différences avec un ordinateur de bureau :
 - L'interface IHM peut être aussi simple qu'une led qui clignote ou aussi complexe qu'un système de vision de nuit en Temps Réel.
 - Des circuits numériques FPGA, ASIC ou des circuits analogiques sont utilisés en plus pour augmenter les performances du système ou sa fiabilité.
 - Le logiciel a une fonctionnalité fixe à exécuter et est spécifique à une application.

LES 4 TYPES DE SYSTEMES EMBARQUES

General Computing

- Application similaire à une application de bureau mais empaquetée dans un système embarqué.
- jeu vidéo, set- top box.

Control Systems

- Contrôle de systèmes en Temps Réel.
- Moteur d'automobile, process chimique, process nucléaire, système de navigation aérien.

Signal Processing

- Calcul sur de grosses quantités de données.
- Radar, Sonar, compression vidéo.

Communication & Networking

- Transmission d'information et commutation.
- Téléphone, Internet.

EXEMPLE : WIRELESS

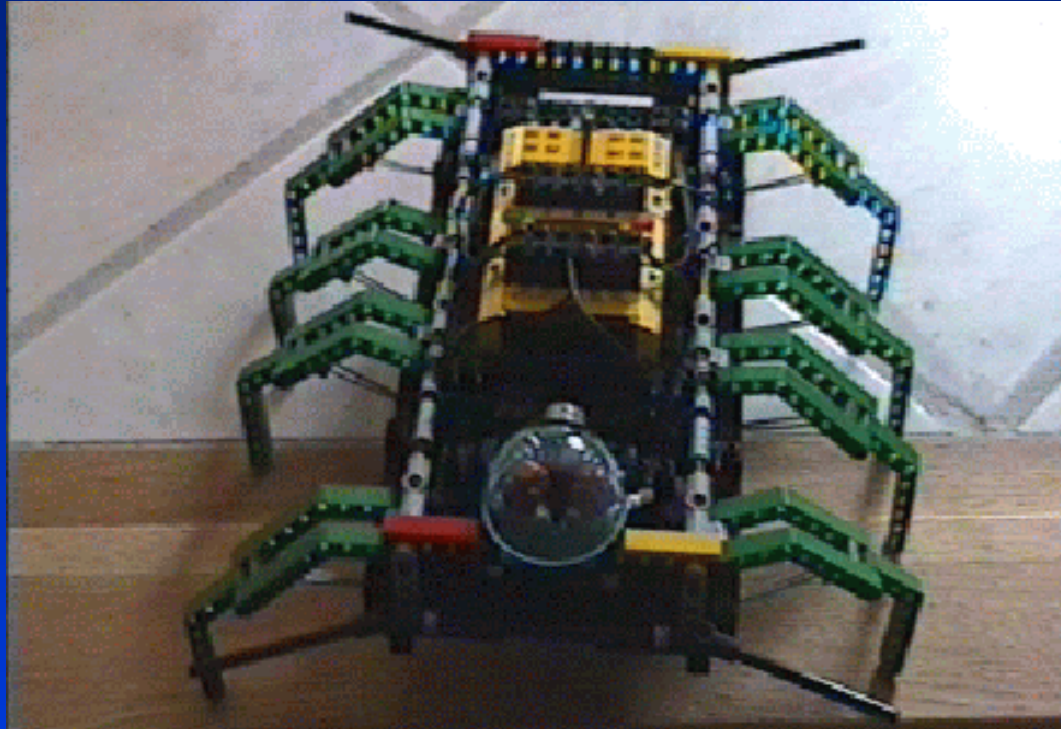


Hand-held GPS Units



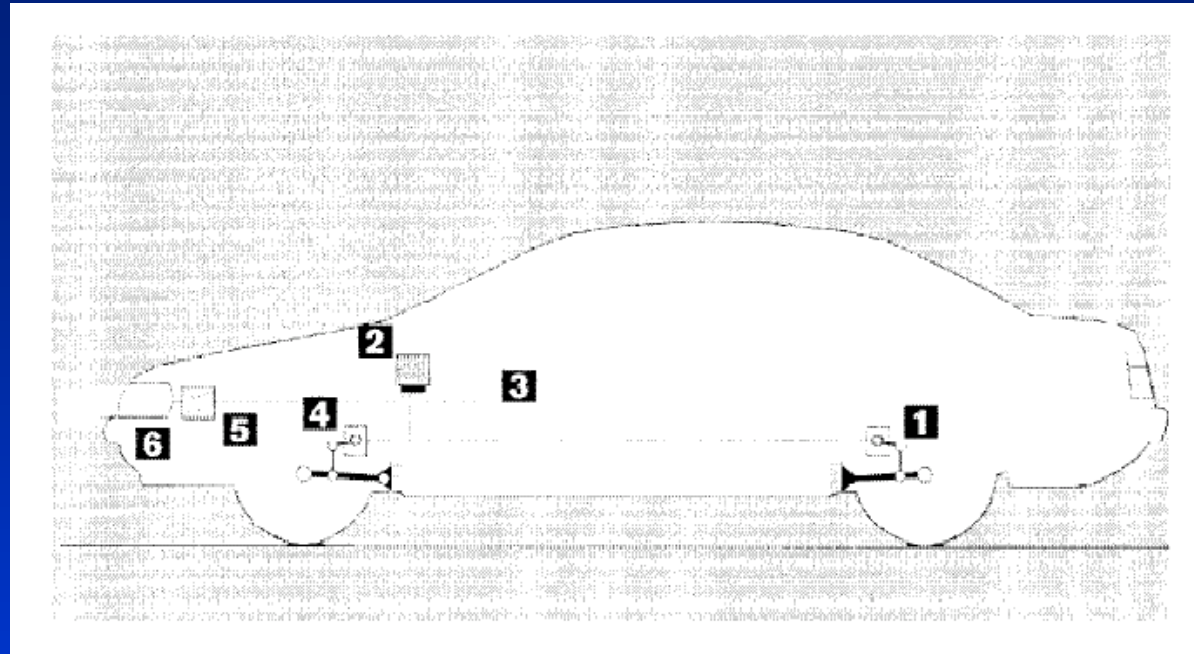
Telematics System for Automobiles

EXEMPLE : ROBOTIQUE



Spider robot – constructed with LEGO Mindstorms Components

EXEMPLE : AUTOMOBILE



Car with an automatic headlight leveling system. 1: Rear distance Sensor, 2: Control unit, 3: Speed signal, 4: Front distance sensor, 5: Motor, 6: Lamps.

CARACTERISTIQUES D 'UN SYSTEME EMBARQUE

- Fonctionnement en Temps Réel :
 - Réactivité : des opérations de calcul doivent être faites en réponse à un événement extérieur (interruption matérielle).
 - La validité d 'un résultat (et sa pertinence) dépend du moment où il est délivré.
 - Rater une échéance va causer une erreur de fonctionnement.
 - Temps Réel dur : plantage.
 - Temps Réel mou : dégradation non dramatique des performances du système.
 - Beaucoup de systèmes sont « multirate » : traitement d 'informations à différents rythmes.

CARACTERISTIQUES D 'UN SYSTEME EMBARQUE

- Faible encombrement, faible poids :
 - Electronique « *pocket PC* », applications portables où l 'on doit minimiser la consommation électrique (bioinstrumentation...).
 - Difficulté pour réaliser le packaging afin de faire cohabiter sur une faible surface électronique analogique, électronique numérique, RF sans interférences.
- Faible consommation :
 - Batterie de 8 heures et plus (PC portable : 2 heures).

CARACTERISTIQUES D 'UN SYSTEME EMBARQUE

- Environnement :
 - Température, vibrations, chocs, variations d 'alimentation, interférences RF, corrosion, eau, feu, radiations.
 - Le système n 'évolue pas dans un environnement contrôlé.
 - Prise en compte des évolutions des caractéristiques des composants en fonction de la température, des radiations...

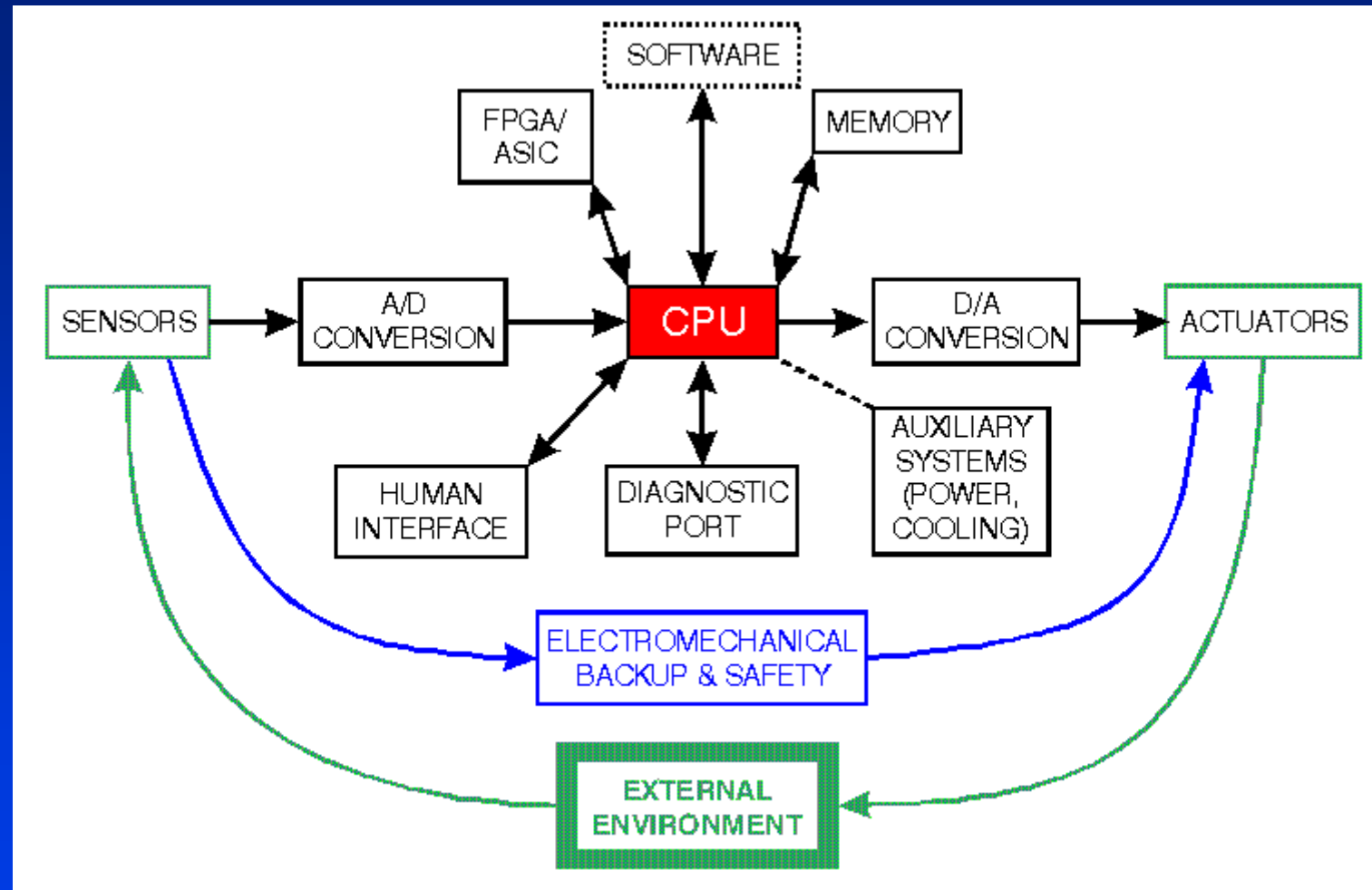
CARACTERISTIQUES D 'UN SYSTEME EMBARQUE

- Fonctionnement critique pour la sécurité des personnes. Sûreté :
 - Le système doit toujours fonctionner correctement.
 - Sûreté à faible coût avec une redondance minimale.
 - Sûreté de fonctionnement du logiciel
 - Système opérationnel même quand un composant électronique lâche.
 - Choix entre un design tout électronique ou électromécanique.

CARACTERISTIQUES D 'UN SYSTEME EMBARQUE

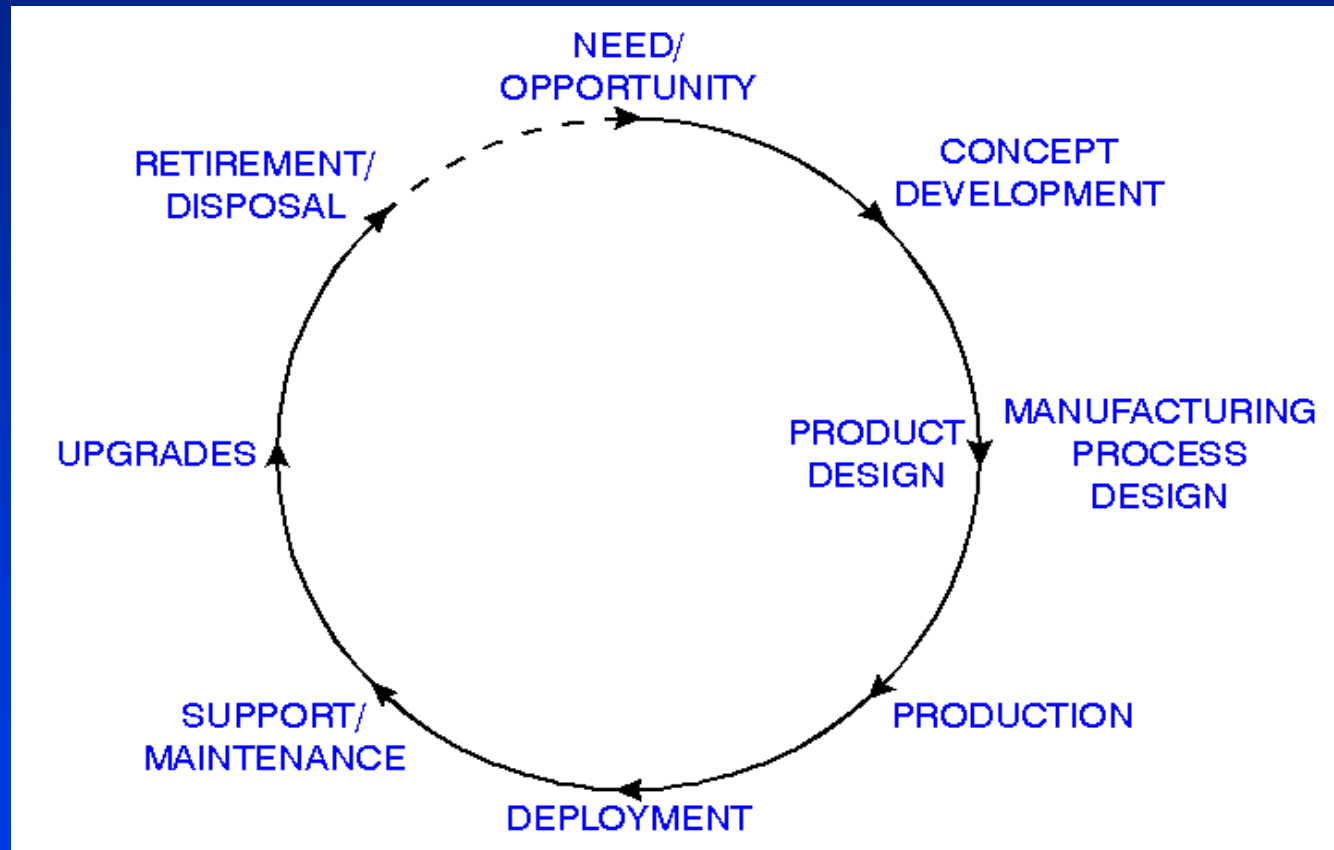
- Beaucoup de systèmes embarqués sont fabriqués en grande série et doivent avoir des prix de revient extrêmement faibles, ce qui induit :
 - Une faible capacité mémoire.
 - Un petit processeur (4 bits). Petit mais en grand nombre !
- La consommation est un point critique pour les systèmes avec autonomie.
 - Une consommation excessive augmente le prix de revient du système embarqué car il faut alors des batteries de forte capacité.
- Faible coût :
 - Optimisation du prix de revient.

SYSTEME EMBARQUE TYPIQUE



CARACTERISTIQUES D 'UN SYSTEME EMBARQUE

- Cycle de vie d'un système embarqué :



LES SYSTEMES EMBARQUES ET LE TEMPS REEL

- Généralement, un système embarqué doit respecter :
 - des contraintes temporelles fortes (*Hard Real Time*).
 - on y trouve enfoui un système d'exploitation ou un noyau Temps Réel (*Real Time Operating System, RTOS*).
- Le Temps Réel est un concept un peu vague. On pourrait le définir comme : "*Un système est dit Temps Réel lorsque l'information après acquisition et traitement reste encore pertinente*".

LES SYSTEMES EMBARQUES ET LE TEMPS REEL

- Cela veut dire que dans le cas d'une information arrivant de façon périodique (sous forme d'une interruption périodique du système), les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information.
- Pour cela, il faut que le noyau ou le système Temps Réel soit *déterministe* et *préemptif* pour toujours donner la main durant le prochain *tick* à la tâche de plus forte priorité prête.

LES SYSTEMES EMBARQUES ET LE TEMPS REEL

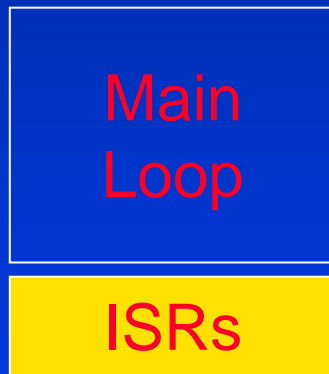
- Une confusion classique est de mélanger Temps Réel et rapidité de calcul du système donc puissance du processeur (microprocesseur, microcontrôleur, DSP).
- On entend souvent : “ *Être temps Réel, c’est avoir beaucoup de puissance : des MIPS, des MFLOPS...* ”.

LES (RT)OS ET LES SYSTEMES EMBARQUES AUJOURD 'HUI

- La question d'utiliser un système d'exploitation Temps Réel ou non ne se pose plus aujourd'hui pour des raisons évidentes :
 - Simplification de l'écriture de l'application embarquée.
 - Portabilité.
 - Evolutivité.
 - Maîtrise des coûts.
 - ...
- Le système d'exploitation peut être même maison : encore dans 50 % des cas !

LES (RT)OS ET LES SYSTEMES EMBARQUES AUJOURD 'HUI

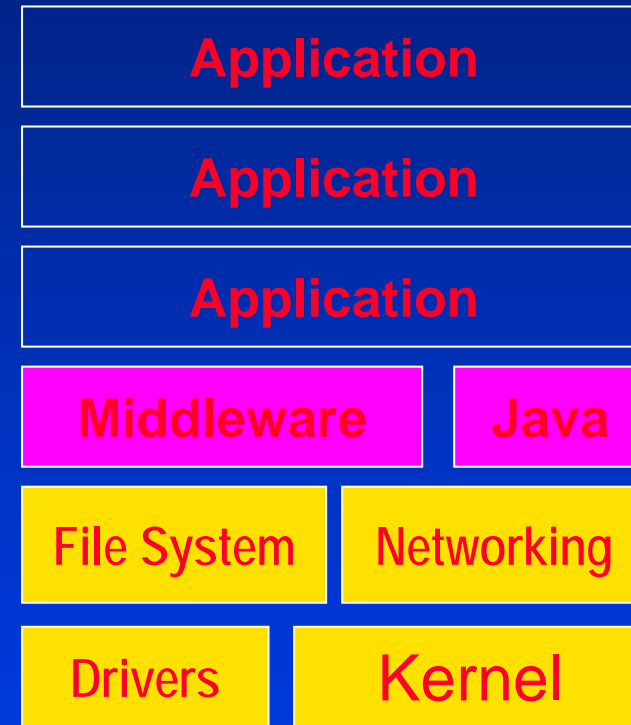
superboucle



Années 70



Années 80-90



Aujourd'hui

LES (RT)OS ET LES SYSTEMES EMBARQUES AUJOURD 'HUI

- La superboucle n 'est pas périmée mais reste réservée aux petits systèmes (8 bits).
- Le choix doit être bien sûr le bon choix pour minimiser les coûts du système. On n 'oublie pas les bonnes recettes du passé !

LES SYSTEMES EMBARQUES ET LINUX

- Linux depuis 2001 est en train de conquérir un domaine où on ne l'attendait pas vraiment : l'univers des systèmes embarqués.
- Pourquoi retrouve-t-on Linux dans l'embarqué ? Tout d'abord pour ses qualités qu'on lui reconnaît maintenant dans l'environnement plus standard du PC grand public :
 - Libre, disponible gratuitement au niveau source : pas de royalties à reverser.
 - Ouvert.

LES SYSTEMES EMBARQUES ET LINUX

- Pourquoi retrouve-t-on Linux dans l'embarqué ? Tout d'abord pour ses qualités qu'on lui reconnaît maintenant dans l'environnement plus standard du PC grand public :
 - Stable et efficace.
 - Aide rapide en cas de problèmes par la communauté Internet des développeurs Linux.
 - Nombre de plus en plus important de logiciels disponibles.
 - **Connectivité IP en standard.**

LES SYSTEMES EMBARQUES ET LINUX

- Linux a aussi d'autres atouts très importants pour les systèmes embarqués :
 - Portage sur processeurs autres que x86 : PowerPC, ARM, MIPS, 68K, ColdFire...
 - Taille du noyau modeste compatible avec les tailles de mémoires utilisées dans un système embarqué (<500 Ko).
 - Différentes distributions proposées suivant le domaine : routeur IP, PDA, téléphone...
 - Support du chargement dynamique de modules qui permet d'optimiser la taille du noyau.
 - Migration rapide et en douceur pour un spécialiste Linux à Linux embarqué ; ce qui réduit les temps de formation (et les coûts...).

LES SYSTEMES EMBARQUES ET LINUX

- On a en fait entendu parler pour la première fois officiellement de Linux embarqué à une exposition *Linux World* en 1999 où les sociétés Motorola, Force et Ziatech ont présenté un système CompactPCI fonctionnant sous Linux.
- En 2000 a été créé le consortium Linux embarqué (*Embedded Linux Consortium*) dont le but est de centraliser et de promouvoir les développements de solutions Linux embarqué. Ce consortium regroupe des éditeurs de distribution Linux, des éditeurs de systèmes Temps Réel propriétaires (comme WindRiver pour VxWorks) et des fabricants de composants. Il compte actuellement plus de 100 membres.

LES SYSTEMES EMBARQUES ET LINUX

- Les distributions Linux embarqué ont une part de marché grandissante face à des distributions propriétaires généralement Temps Réel comme VxWorks, pSOS, QNX... où l'on est d'abord obligé de payer pour accéder à la plateforme de développement puis de payer des royalties pour chaque système (ou cible) que l'on commercialise ensuite.
- Il est à noter que l'on observe une évolution de ce système à péage de certains face à la “ menace ” Linux.

LES SYSTEMES EMBARQUES ET LINUX

- Linux embarqué supporte aussi différentes extensions Temps Réel qui mettent en place une couche d'abstraction logique entre matériel, interruptions et Linux. Linux et l'ensemble des processus sont généralement considérés comme la tâche de fond exécutée quand il y a rien de Temps Réel à faire...
- On peut citer comme extensions Temps Réel :
 - Xenomai.
 - RTAI.
 - RTLinux racheté par WindRiver/Intel. Plus supporté ?

CHAPITRE 2 : IMPORTANCE DU CODESIGN DANS L'EMBARQUE

CODESIGN : QUAND LE MATERIEL REJOINT LE LOGICIEL

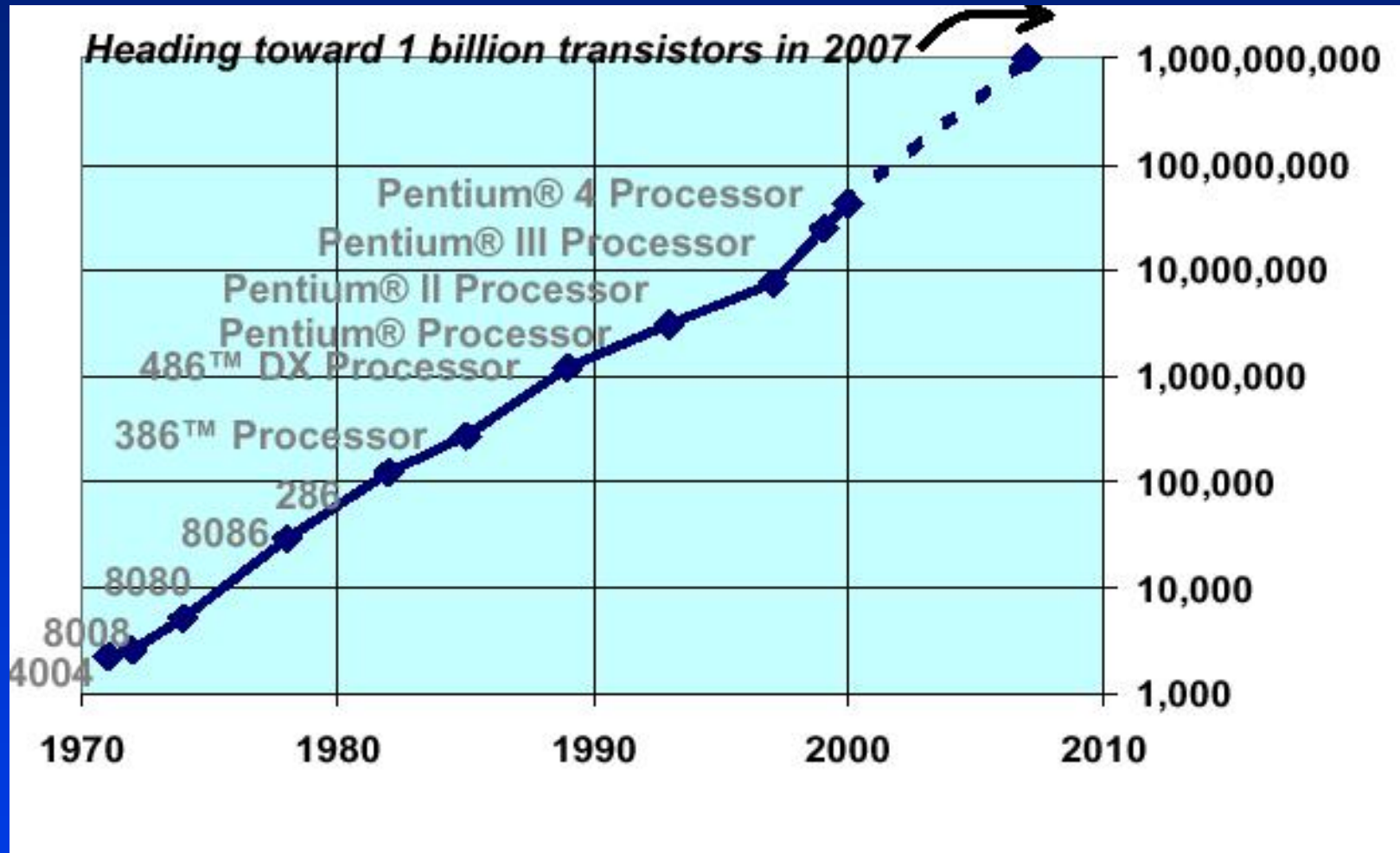
- La capacité de conception de systèmes numériques permet aujourd'hui de tout intégrer dans un même composant (concept du *single chip*).
- On travaille donc au niveau système et non plus au niveau porte élémentaire ou schématique. On parle de système sur silicium SoC (*System on Chip*) ou SoPC (*System on Programmable Chip*).
- Ceci est lié à la loi empirique de Moore qui dit que pour une surface de silicium donné, on double le nombre de transistors intégrés tous les 18 mois !

CODESIGN : QUAND LE MATERIEL REJOINT LE LOGICIEL

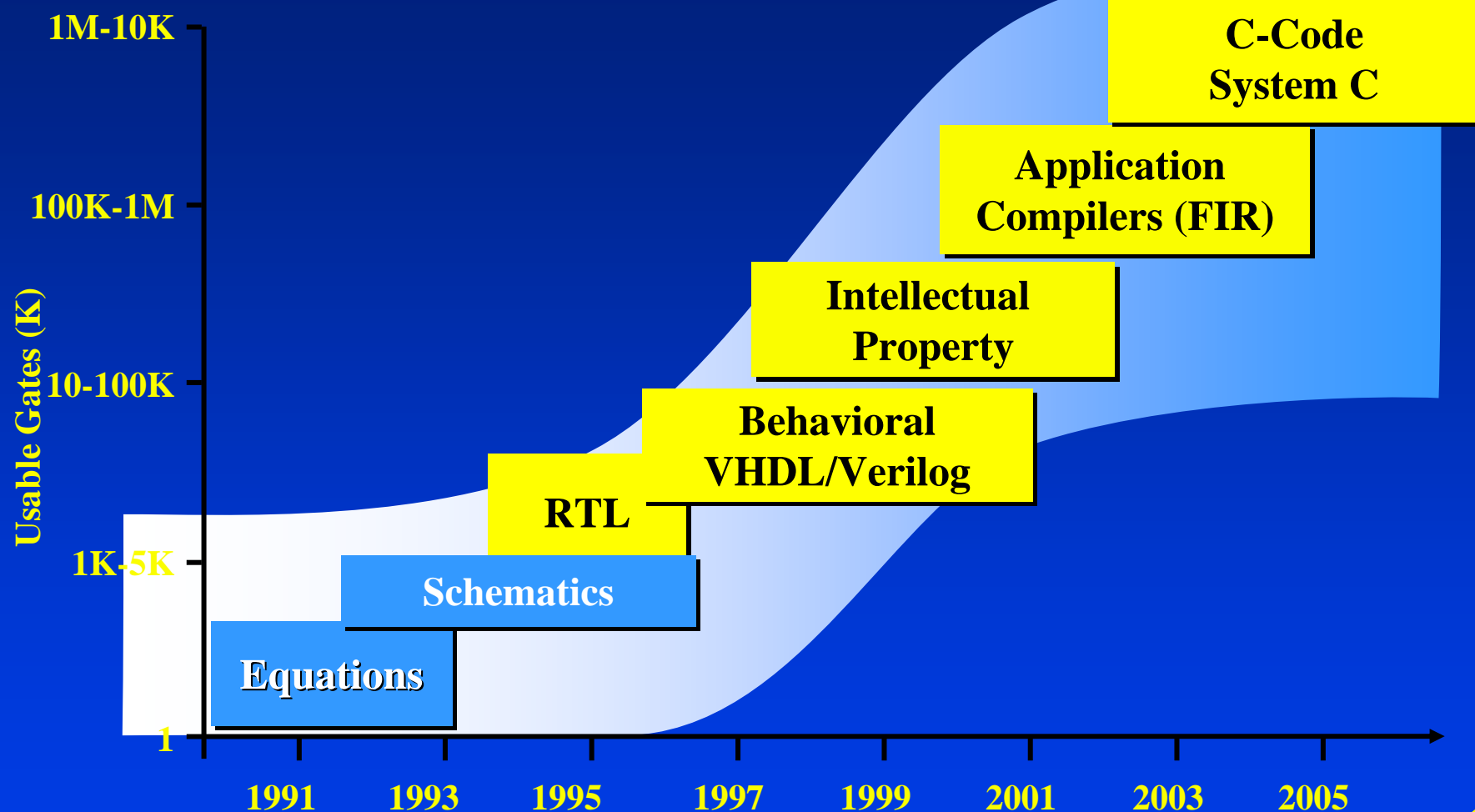
	1998	1999	2001	2009
Technologie	0,25 μm	0,18 μm	0,15 μm	40 nm
Complexité	1 M de portes	2-5 M	5-10 M	1 G
Loi de Moore				



CODESIGN : QUAND LE MATERIEL REJOINT LE LOGICIEL



CODESIGN : QUAND LE MATERIEL REJOINT LE LOGICIEL



CODESIGN : QUAND LE MATERIEL REJOINT LE LOGICIEL

- On utilise maintenant des langages de description du matériel (VHDL, Verilog) pour synthétiser et aussi tester les circuits numériques. On a ainsi une approche logicielle pour concevoir du matériel.
- Avec l'augmentation de l'intégration, les systèmes numériques se sont complexifiés alors que la mise sur le marché doit être la plus rapide possible :
 - Prise en compte du *Time To Market* (TTM).
 - Réutilisation de choses déjà réalisées (*Design Reuse*).

CODESIGN : QUAND LE MATERIEL REJOINT LE LOGICIEL

- On a ainsi vu apparaître la notion de blocs IP (*Intellectual Property*) qui est possible par l'utilisation des langages de description du matériel.
- On achète des blocs IP comme on achète un circuit intégré :
 - interface CAN.
 - DCT.
 - Interface MAC IEEE 802.3 10BaseT qui est la condition nécessaire pour assurer la connectivité IP sur réseau Ethernet.

HARDWARE OU *software* ?

- La difficulté est maintenant de savoir comment implémenter une fonctionnalité. Exemple d'un algorithme de compression vidéo :
 - Plus rapide par hardware mais plus cher.
 - Plus flexible par logiciel mais plus lent.
- On doit être capable de jongler en plus avec les paramètres suivants :
 - Coût.
 - Rapidité.
 - Robustesse.

APPROCHE TRADITIONNELLE DE DEVELOPPEMENT D 'UN SYSTEME EMBARQUE

- 1. Choix du matériel (composants électroniques, processeur...) pour le système embarqué.
- 2. Donner le système ainsi conçu aux programmeurs.
- 3. Les programmeurs doivent réaliser un logiciel qui « colle » au matériel en n 'exploitant que les ressources offertes.

COMPLEXITE GRANDISSANTE D 'UN SYSTEME EMBARQUE

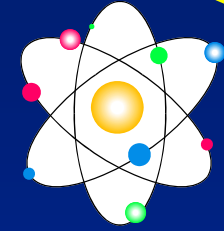
- Les systèmes embarqués sont de plus en plus complexes.
- Il est de plus en plus difficile de penser à une solution globale optimisée du premier jet.
- Il est de plus en plus difficile de corriger les « bugs ».
- Il est de plus en plus difficile de maintenir le système au cours du temps (obsolescence des composants...).
- En conséquence, l 'approche traditionnelle de développement d 'un système embarqué doit évoluer...

MOINS DE TEMPS POUR LE DEVELOPPEMENT

- Dans l'électronique embarquée, le temps de conception devient de plus en plus réduit (pour diminuer les coûts, gagner un appel d'offre).
 - Le temps développement de l'électronique dans l'industrie automobile est passée de 3 à 5 ans à 1 à 3 ans.
- Et le système doit toujours être toujours aussi sûr et robuste.



SOLUTION A LA COMPLEXITE



- Dans le processus de conception du système, on doit garder un niveau d'abstraction important le plus longtemps possible.
- Le système doit pouvoir être décomposé en sous-systèmes suivant une hiérarchie logique (approche objet).
 - Si le design change, on doit pouvoir en réutiliser une bonne partie (*design reuse*).
 - Il ne faut pas jeter à la poubelle un précédent design et repartir «*from scratch*» !
- On doit pouvoir utiliser des outils de vérification automatique.

L 'INTERET DE L 'EMPLOI D 'UN PROCESSEUR

- L 'usage d 'un processeur facilite la réalisation du système dans la majorité des cas.
- Il se peut aussi que le processeur soit surdimensionné par rapport à la fonctionnalité logique à implémenter par logiciel. L 'usage de circuits logiques programmables FPGA (ou des ASIC) est alors intéressante dans ce cas.
- L 'usage d 'un processeur spécialisé comme un microcontrôleur ou un DSP peut aussi réduire considérablement le surdimensionnement et le nombre de composants électroniques.

LA QUADRATURE DU CERCLE

- Quand on conçoit un système embarqué, un certain nombre de contraintes apparaissent :
 - Quel hardware a-t-on besoin ?
 - Quel processeur choisir ? Quelle puissance CPU ?
 - Quelle type de mémoire et taille mémoire a-t-on besoin ?
 - Choix entre hardware rapide ou software intelligent ?
 - La consommation ? Minimiser les accès mémoire ? Choisir les instructions assembleur en fonction de leur consommation ?
 - Les contraintes de temps de conception seront-elles respectées ?
Le TTM ?

LA QUADRATURE DU CERCLE

- Quand on conçoit un système embarqué, un certain nombre de contraintes apparaissent :
 - Est-ce que le système final marche correctement ?
 - Est-ce que les spécifications fonctionnelles ont été respectées ?
 - Comment doit-on tester les caractéristiques Temps Réel du système ? Doit-on le tester avec des données réelles ? Quelle plateforme de tests doit-on utiliser ?

LA QUADRATURE DU CERCLE ET LE CONCEPTEUR DE SYSTEME EMBARQUE

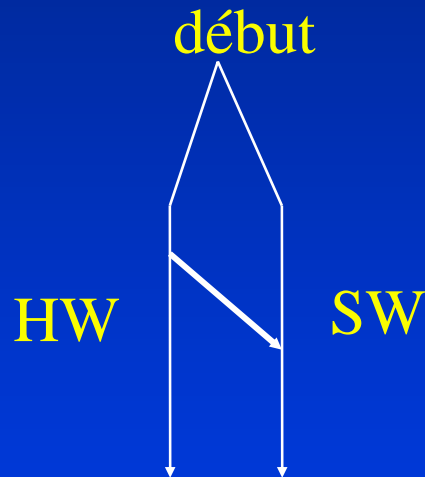
- Concevoir un système embarqué remplissant toutes les contraintes précédemment énoncées demande au concepteur des compétences multidisciplinaires :
 - Compétences hardware et software : microprocesseur, microcontrôleur, DSP, FPGA, codesign, VDHL, assembleur, E/S, C, RTOS, réseau, UML, Linux, Java...
 - Connaissance des systèmes numériques.
 - Savoir travailler en équipe avec des ingénieurs d'autres disciplines.
 - Comprendre le besoin du client et savoir aussi l'identifier !

CODESIGN HARDWARE/SOFTWARE

- Le codesign dans la méthodologie de conception d'un système embarqué est de plus en plus utilisé.
- Le codesign permet de concevoir en même temps à la fois le matériel et le logiciel pour une fonctionnalité à implémenter. Cela est maintenant possible avec les niveaux d'intégration offerts dans les circuits logiques programmables.
- Le codesign permet de repousser le plus loin possible dans la conception du système les choix matériels à faire contrairement à l'approche classique où les choix matériels sont faits en premier lieu !

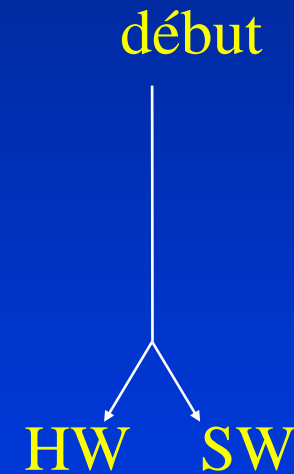
CONCEPTION ET CODESIGN

Conception traditionnelle



Réalisée par des groupes d'ingénieurs indépendants

Codesign (flot concurrent)

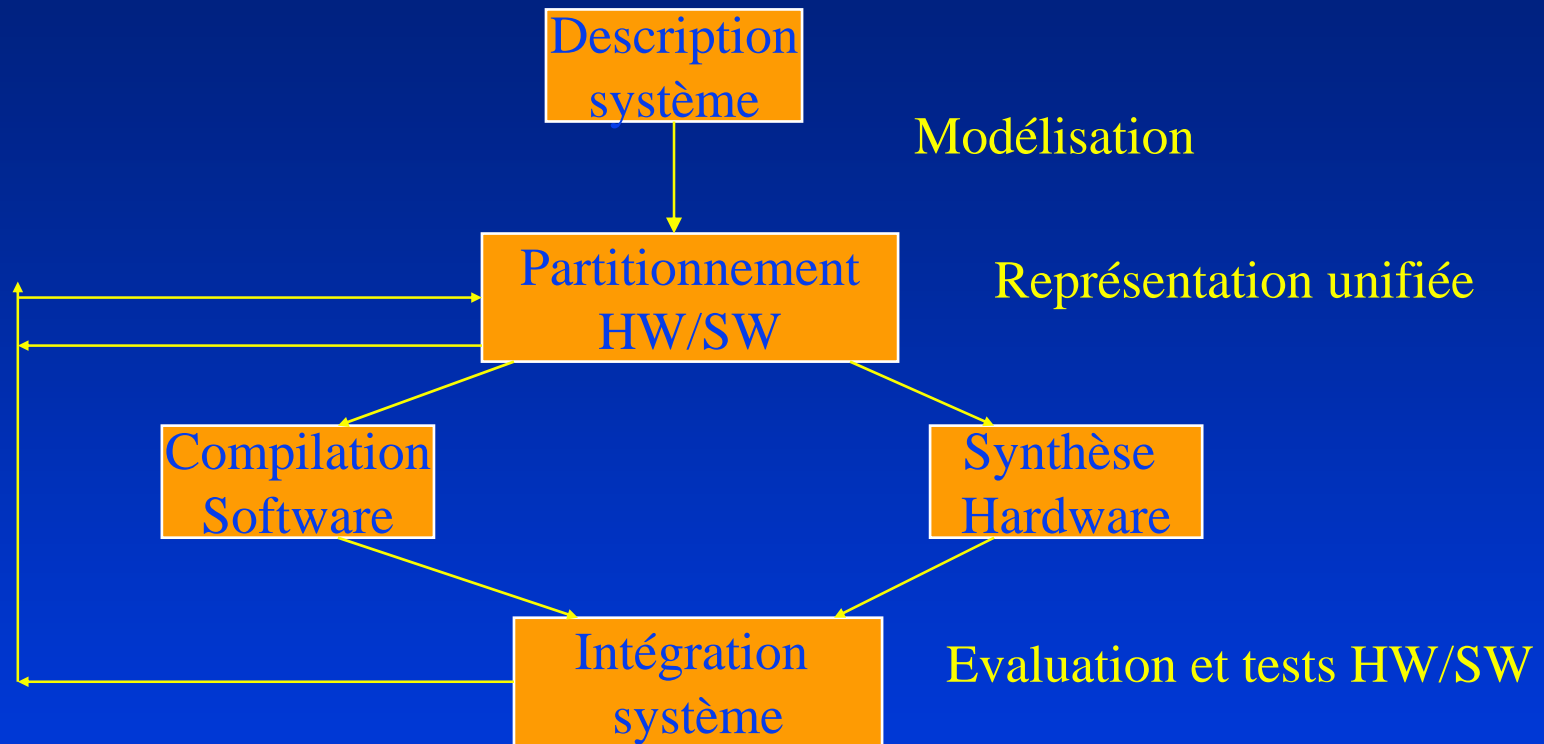


Réalisée par le même groupe d'ingénieurs en coopération

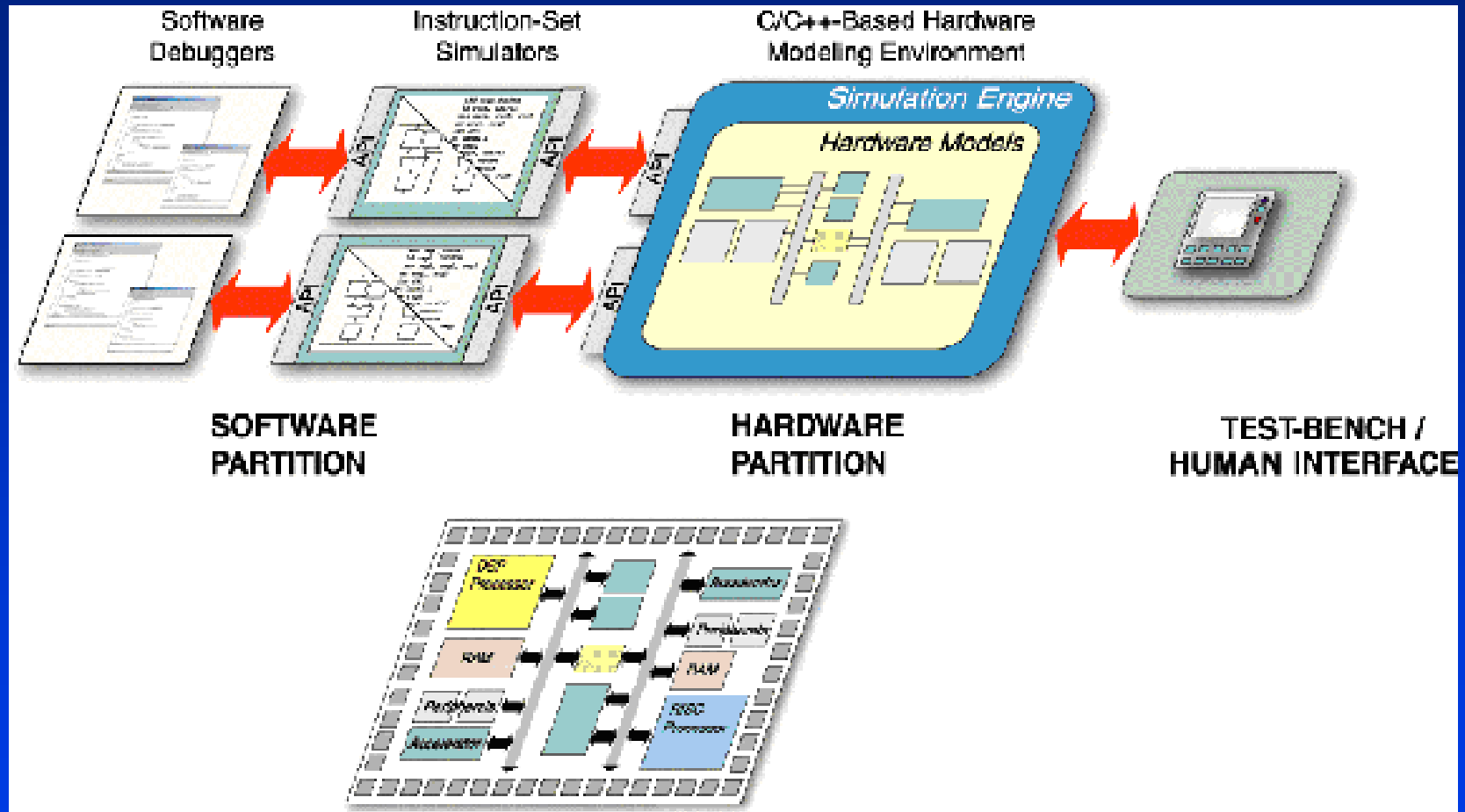
LES ETAPES DANS LE CODESIGN

- Durant le processus de codesign, on distingue les étapes suivantes :
 - Spécifications : liste des fonctionnalités du système de façon abstraite.
 - Modélisation : conceptualisation et affinement des spécifications produisant un modèle du matériel et du logiciel.
 - Partitionnement.
 - Synthèse et optimisation : synthèse matérielle et compilation logicielle.
 - Validation : cosimulation.
 - Intégration.
 - Tests d'intégration.

LES ETAPES DANS LE CODESIGN



CODESIGN HARDWARE/SOFTWARE



AVANTAGES DU CODESIGN

- Le codesign est intéressant pour la conception de systèmes embarqués (ou de SoC) :
 - Amélioration des performances : parallélisme, algorithmes distribués, architecture spécialisée...
 - **Reconfiguration** statique ou dynamique en cours de fonctionnement.
 - Indépendance vis à vis des évolutions technologiques des circuits logiques programmables.
 - Mise à profit des améliorations des outils de conception fournis par les fabricants de circuits logiques programmables : synthèse plus efficace, performance accrue.

CODESIGN ET SYSTEMES EMBARQUES : UN BILAN

- Les systèmes embarqués étant de plus en plus complexes, une méthodologie rigoureuse de conception doit être maintenant mise en œuvre.
- Le codesign fait partie intégrante de cette méthodologie. Cela est maintenant possible grâce à des niveaux d'intégration sur silicium très importants des circuits logiques programmables qui peuvent aujourd'hui embarquer des processeurs.
- L'apparition de modules IP, véritables circuits électroniques sous forme logicielle (« le *java bean* de l'électronique ») n'a fait qu'accentuer l'importance du codesign et du *design reuse*.

CHAPITRE 3 :

LA CONCEPTION ET LA MISE AU POINT DES SYSTEMES EMBARQUES

PARTIE 1 : INTRODUCTION

INTRODUCTION

- La conception et la mise au point d'un système embarqué est un processus difficile où le concepteur est vite en proie au doute et demande un certain talent.
- Ce chapitre se propose de donner quelques pistes et quelques règles de bon sens dans l'art de mettre au point (« debugger ») un système embarqué :
 - Conception : mise au point du matériel.
 - Tests : mise au point du logiciel avec éventuellement modification du matériel.
- L'expérience acquise lors de la mise au point de précédents systèmes est un capital inestimable...

INTRODUCTION

- La mise au point (debug) d'un système est pénible, long et consommateur d'argent.
- Dans la conception matérielle d'un circuit SoC, cette étape de tests/debug (avec des patterns bien choisis) consomme 60 % du temps du projet.
- Du point de vue logiciel, on estime qu'il reste encore 5 % de bugs après compilation statique d'un code source. Sur un programme de 10000 lignes, cela représente 500 bugs avant inspection du code et tests. L'inspection du code permet d'en éliminer 70 à 80 %, ce qui en laisse encore 100. Après tests, on estime en avoir éliminé 50 %, ce qui en laisse 50 dans le produit fini ! (d'après J. Ganssle)

INTRODUCTION

- Le calcul précédent est intolérable bien sûr (à moins de vouloir vendre des releases à tout coup) et l'on a donc besoin d'outils très performants et chers (analyseur de couverture du code...) pour faire tendre ce chiffre vers 0.
- Toutes les techniques de management et méthodologie n'élimineront jamais le besoin de tester et de « debugger »...
- Le debug est ainsi très important pour ne pas « surinfecter » le produit fini...

PARTIE 2 :

LES OUTILS POUR LA MISE AU POINT D 'UN SYSTEME EMBARQUE

INTRODUCTION

- Dans le processus de mise au point du système électronique, il est primordial de bien connaître la palette d'outils à disposition.
- Mettre au point un système est d'autant plus compliqué que l'on travaille avec un matériel non encore testé...
- A chaque problème, il existe l'outil approprié...il convient donc de les présenter...

L'OSCILLOSCOPE

INTRODUCTION

- L'oscilloscope est le premier outil de debug hard.
- Il permet de visualiser réellement un signal électrique :
 - Analogique.
 - Digitale (sans niveaux logiques comme avec l'analyseur logique).

AVANTAGES ET INCONVENIENTS

- L'oscilloscope permet de mettre en évidence parasites, glitches...
- Il est limité à 4 sondes au plus soit 4 signaux.
- La logique de synchronisation (trigger) est très limitée.
- Il peut être intrusif en rajoutant une charge :
 - un circuit logique marche quand on branche une sonde sur une broche puis ne marche plus quand on la débranche (charge capacitive).
- Il est important d'avoir de bonnes sondes **calibrées** avec une grande bande passante pour ne pas déformer le signal.

L'ANALYSEUR LOGIQUE

INTRODUCTION

- L'analyseur logique espionne des signaux logiques : bus d'adresses et de données, signaux de contrôle...
- On peut collecter les accès du processeur en mémoire et ensuite désassembler le code exécuté.
- On voit des adresses physiques externes donc si le cache d'instructions ou de données est activé, on ne voit pas tout...
- Si les caches du processeur sont désactivés, il faudra faire attention au *prefetch* du processeur.

AVANTAGES

- L 'analyseur logique est un outil universel dans la mise au point de systèmes numériques. Il n 'est pas lié à un processeur particulier.
- L 'analyseur logique permet de mettre en place des triggers de déclenchement de l 'acquisition.
- Il est important de garder à l 'esprit (idem pour l 'émulateur ICE) que dans 90 % des cas, on choisit des conditions de trigger simples (booléen) par rapport aux triggers complexes tant vantés par les constructeurs ; ce qui en augmente le prix !

INCONVENIENTS

- Un analyseur logique est cher.
- Il y a le problème de la connectique et du nombre important de signaux à capturer.
- Pour désassembler le code exécuté, on est obligé de capturer les bus d'adresses et de données ainsi que les signaux de contrôle soit un nombre très important de signaux logiques.
- Si l'on a validé l'optimisation du code (compilé ou assemblé), il peut être difficile de remonter au fichier source par désassemblage.

INCONVENIENTS

- Quand on branche l'analyseur logique, on introduit une charge différente sur un bus. On peut ainsi masquer ou créer des problèmes.
- La charge peut réduire le bruit induit et l'ensemble marche dans un environnement bruité. On enlève l'analyseur et plus rien ne marche. Cela peut être aussi le contraire.
- L'analyseur logique est donc intrusif.
- L'analyseur logique est passif : on n'a pas de contrôle sur le processeur.

INCONVENIENTS

- Il faudra faire attention aux caches validés.
- L'analyseur logique fait un échantillonnage. Il convient donc d'avoir une fréquence d'acquisition suffisante et une taille mémoire d'acquisition conséquente...

LE MONITEUR ROM

INTRODUCTION

- Le moniteur ROM est un programme de debug accessible depuis un port de communication (liaison série).
- Il dialogue avec une application debugger pour lui envoyer la valeur courante d'un registre, d'une adresse mémoire et placer un point d'arrêt simple à une adresse donnée ou télécharger en mémoire du code.

POINT D'ARRÊT

- Un point d'arrêt (breakpoint) permet de dérouter l'exécution normale du code de l'application du système pour en examiner l'état (du processeur, de la mémoire, des E/S...) ou le modifier.
- L'instruction pointée par le point d'arrêt (son adresse) est remplacée par une routine spécifique (ou un *trap*) :
 - Sauvegarde des registres du processeur.
 - Examen de l'état du processeur.
 - Restauration des registres du processeur.

POINT D'ARRET

Code image in memory: before

.....
.....
instruction n-1
instruction n
instruction n+1
.....
.....

Want to set
breakpoint here

Debugger inserts
trap vector in place
of instruction

Code image in memory: after

.....
.....
instruction n-1
trap vector to debugger entry point
instruction n+1
.....
.....
instruction n

Moved to the debugger
database for safe keeping

EXEMPLE DE MONITEUR

- Moniteur Buffalo pour le microcontrôleur 68HC11 :

```
BUFFALO 3.4 Ex (ENSEIRB) - Bit User Fast Friendly Aid to Logical Operation

68HC11E9 CPU
8K BUFFALO MONITOR PROGRAM EPROM: $8000 TO $9FFF
DEFAULT INTERNAL RAM & REGISTER ALLOCATION
EEPROM: $B600 TO $B7FF
EXTERNAL RAM 32K : $0000 TO $7FFF
>

ASM [<addr>] Line asm/disasm
  [/,=] Same addr,      [^,-] Prev addr,      [+ ,CTLJ] Next addr
  [CR] Next opcode,    [CTLA,..] Quit
BF <addr1> <addr2> [<data>] Block fill memory
BR [-][<addr>] Set up bkpt table
BULK Erase EEPROM,          BULKALL Erase EEPROM and CONFIG
CALL [<addr>] Call subroutine
GO [<addr>] Execute code at addr,          PROCEED Continue execution
EEMOD [<addr> [<addr>]] Modify EEPROM range
LOAD, VERIFY [T] <host dwnld command> Load or verify S-records
MD [<addr1> [<addr2>]] Memory dump
MM [<addr>] or [<addr>]/ Memory Modify
  [/,=] Same addr,  [^,-,CTLH] Prev addr,  [+ ,CTLJ,SPACE] Next addr
  <addr>0 Compute offset,          [CR] Quit
MOVE <s1> <s2> [<d>] Block move
OFFSET [-]<arg> Offset for download
RM [P,Y,X,A,B,C,S] Register modify
STOPAT <addr> Trace until addr
T [<n>] Trace n instructions
TM Transparent mode (CTLA = exit, CTLE = send brk)
[CTLW] Wait,          [CTLX,DEL] Abort          [CR] Repeat last cmd
>
```

00:02:37 connecté ANSI 9600 8-N-1 Défil Maj Num Capturer Imprimer l'écho

AVANTAGES

- Le moniteur ROM est bon marché.
- Il n'y a pas de problèmes de connectique.
- Il n'y a pas de problèmes de rapidité (c'est du code exécuté).

INCONVENIENTS

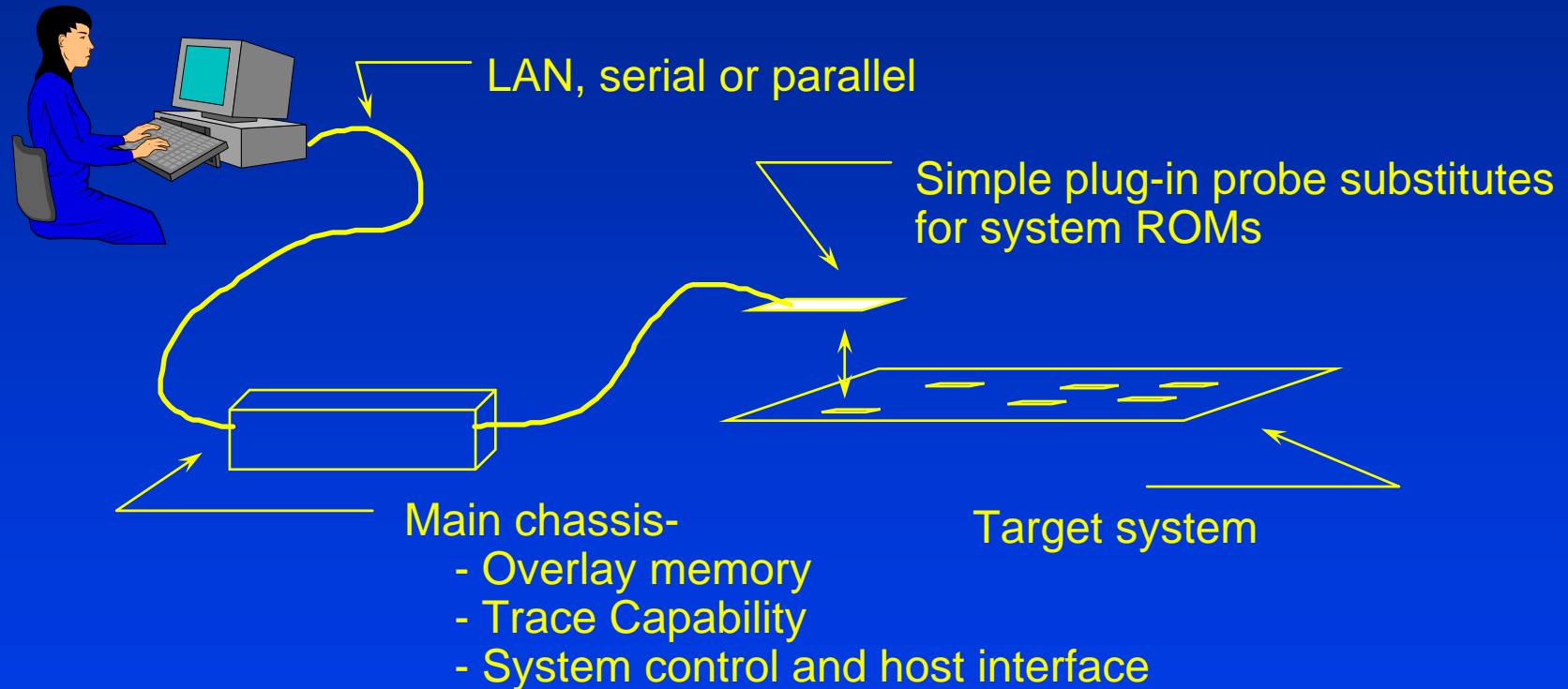
- Le moniteur ROM demande un port de communication exclusif à prévoir dans le design.
- Il consomme des ressources du système cible : RAM, ROM, interruptions ; ce qui peut être gênant pour un tout petit système.
- Il y a toujours des problèmes de configuration lors de la prise en main.
- Il ne marche pas s'il y a des bugs matériels.
- Il ne possède pas ou peu de possibilités de debug en Temps Réel.

L'EMULATEUR ROM

INTRODUCTION

- L'émulateur ROM est un équipement qui se branche sur un support de mémoire ROM du système cible mais contient de la RAM au lieu de la ROM (*Overlay RAM*)
- Il possède :
 - Un câble de liaison en concordance avec les spécificités du support ROM.
 - De la RAM rapide.
 - Un contrôle du processeur (reset)...
 - Un port de communication avec le système hôte pour le debug.
- C'est un moyen simple et rapide de télécharger du code dans la cible en lieu et place de la mémoire ROM initiale.

INTRODUCTION



AVANTAGES

- L 'émulateur ROM est bon marché.
- Il est compatible avec différents types de mémoires et n 'est donc pas lié à un type de processeur comme l 'émulateur ICE
- Il permet d 'ajouter un port de communication temporaire au système.
- Le téléchargement de code est rapide.
- On peut tracer l 'activité ROM en Temps Réel.

AVANTAGES

- On peut mettre un point d'arrêt en ROM.
- Il peut être utilisé avec d'autres outils de debug.

INCONVENIENTS

- L 'émulateur ROM a besoin d 'être de plus en plus rapide (comme l 'émulateur ICE) avec l 'évolution des processeurs.
- Il ne fonctionne qu 'avec une interface ROM : problème si l 'on a un microcontrôleur avec de la ROM interne.
- Beaucoup de systèmes font une copie ROM vers RAM pour des soucis de performance donc l 'émulateur ROM est hors jeu.
- Il est inefficace en cas de problèmes matériels sur le système.
- Il est intrusif.

L'EMULATEUR ICE

INTRODUCTION

- ICE est l 'acronyme de In Circuit Emulator.
- Pourquoi utiliser un émulateur ICE ?
 - Si le hardware du système n 'est pas parfait.
 - C 'est le meilleur outil pour détecter des problèmes matériels mais aussi logiciels.
 - Il n 'utilise pas de ressources du système cible.
 - Il supporte un debug en Temps Réel avec possibilités de trace, triggers...
- C 'est l 'outil parfait pour un intégration conjointe matérielle et logicielle.

INTRODUCTION

- L'émulateur ICE combine à la fois un debugger, un émulateur ROM et un analyseur logique.
- Il se substitue complètement au processeur qu'il va émuler :

INTRODUCTION

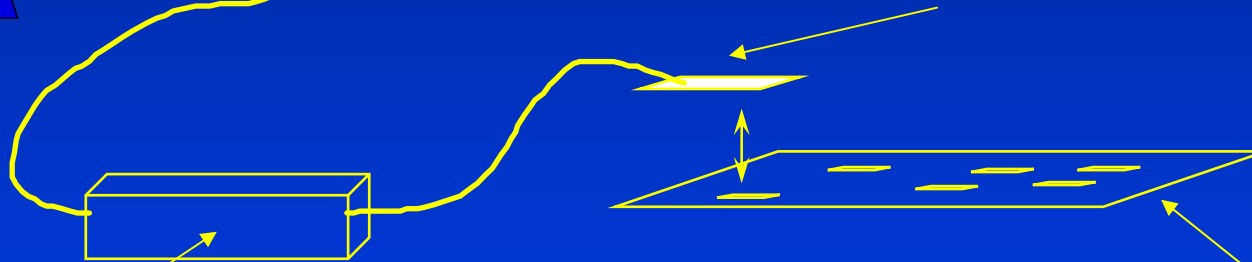
Host computer runs emulator control software

- Provides run control
- Displays real time trace at source level
- Loads overlay memory with object code
- High-speed link to emulation chassis



Probe head contain emulation microprocessor

- Substitutes for, or disables target microprocessor
- Contains run control circuitry and cable buffers
- May contain memory mapping hardware



Main chassis

- Contains emulation (overlay memory)
- Trace analysis hardware and trace memory
- Performance analysis hardware (
- Power supply

Target system

Control and communications

INTRODUCTION

- L'émulateur ICE permet de contrôler le système :
 - Lecture/écriture en mémoire.
 - Modification des registres du processeur.
 - Pas à pas.
 - Points d'arrêt.
 - Désassemblage de code.
 - Téléchargement de code.

INTRODUCTION

- L'émulateur ICE supporte l'Overlay RAM : à l'intérieur de l'émulateur, la RAM est mappée dans l'espace d'adressage du processeur de la cible pour remplacer sa mémoire ROM ou FLASH. On peut ainsi télécharger rapidement du code à tester en Overlay RAM sans reprogrammer la mémoire ROM de la carte cible.
- Point d'arrêt matériel : stoppe l'exécution du programme sans affecter le contexte du processeur.
- Point d'arrêt logiciel : insertion dans le code d'un appel vers une routine de debug. Il y a un problème si le code est en mémoire externe ROM car pas d'écriture possible.

TRACES TEMPS REEL

- L'émulateur ICE permet de tracer l'activité du processeur (comme l'analyseur logique).
- Il admet des triggers complexes pour le lancement d'une acquisition (trace pour la surveillance d'une interruption) et désassemble le code capturé stocké dans une mémoire tampon circulaire.
- Si l'on possède la table des symboles, on peut remonter au fichier source directement.

TRACES TEMPS REEL

Example of Real Time Trace from HP64700 emulator

Trace List		Offset=0		More data off screen (ctrl-F, ctrl-G)		time count	
Label:	Address	Data	Opcode or	Status			
Base:	hex	hex	mnemonic			absolute	
after	004FFA	2700	2700	supr data rd word			
+001	004FFC	0000	0000	supr data rd word		+ 520	nS
+002	004FFE	2000	2000	supr data rd word		+ 1.0	uS
+003	002000	2479	MOVEA.L	0001000,A2		+ 1.5	uS
+004	002002	0000	0000	supr prog		+ 2.0	uS
+005	002004	1000	1000	supr prog		+ 2.5	uS
+006	002006	2679	MOVEA.L	0001004,A3		+ 3.0	uS
+007	001000	0000	0000	supr data rd word		+ 3.5	uS
+008	001002	3000	3000	supr data rd word		+ 4.00	uS
+009	002008	0000	0000	supr prog		+ 4.52	uS
+010	00200A	1004	1004	supr prog		+ 5.00	uS
+011	00200C	14BC	MOVE.B	#000,[A2]		+ 5.52	uS
+012	001004	0000	0000	supr data rd word		+ 6.00	uS
+013	001006	4000	4000	supr data rd word		+ 6.52	uS
+014	00200E	0000	0000	supr prog		+ 7.00	uS

TRACES TEMPS REEL

- The user interface software has access to the linker symbol table
- Can intersperse C or C++ source lines with assembly language execution

```
set source on
display trace mnemonic
```

```
Trace List
Label:      Address      Data      Opcode or Status w/ Source Lines      time
Base:      symbols      hex      mnemonic w/symbols      rel
+009  sysstack:+003FC2  0738  0738  supr data wr word  520
-010  sysstack:+003FC0  0006  0006  supr data wr word  480
+011  main:main+00000A  01AA  01AA  supr prog  520
#####main.c - line 104 #####
      initialize_system();
+012  main:main+00000C  4EB9  JSR   |_initialize_sys  480
+013  main:main+00000E  0000  0000  supr prog  520
+014  main:main+000010  114A  114A  supr prog  480
#####initSystem.c - line 1 thru 38 #####
void refresh_menu_window();

void
initialize_system()
{
+015  |_initialize_sys  4E56  LINK  A6,#00000  520
STATUS:  M68000--Running user program  Emulation trace complete_____.....
```

AVANTAGES

- L'émulateur ICE est l'outil roi pour la mise au point matérielle et logicielle.
- Il fournit toutes les fonctionnalités de debug nécessaires.
- Le contrôle du processeur est garanti quel que soit l'état matériel du système cible.
- Il connaît les instructions en cache du processeur.

INCONVENIENTS

- L 'émulateur ICE coûte très cher.
- Il est fabriqué pour un type de processeur donné. Si l 'on utilise un nouveau processeur, il faudra mettre à jour l 'émulateur ICE.
- Problème de la connectique, du prix et de la fragilité de la tête : adaptation à pourvoir si processeur CMS.
- Il ne supporte pas toujours toutes les variantes d 'un processeur donné (microcontrôleur).
- Les gens pensent que l 'émulateur ICE est difficile à utiliser.

INCONVENIENTS

- Le système cible doit être conçu pour supporter la tête de l'émulateur ICE.
- Un émulateur ICE ne fonctionne pas toujours à la fréquence maximale du processeur.
- Quand on branche l'émulateur ICE, on introduit une charge différente de celle du processeur. On peut masquer ou créer des problèmes.
- La charge introduite peut réduire le bruit induit et l'ensemble marche dans un environnement bruité. On enlève l'émulateur ICE et plus rien ne marche. Cela peut être le contraire.
- L'émulateur ICE est donc intrusif.

JTAG

INTRODUCTION

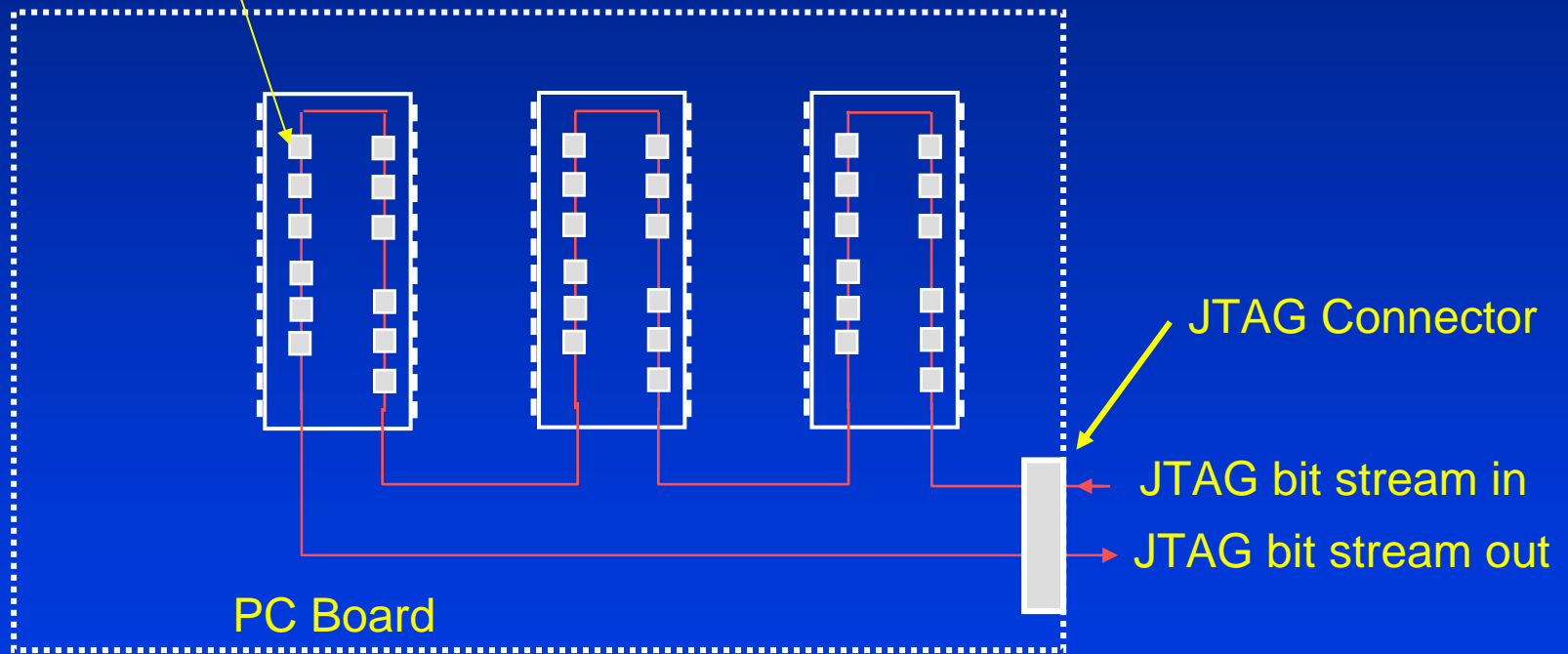
- Le JTAG (Joint Test Action Group) a été originellement créé pour le BIST : Built In Self Test (autotests de circuits électroniques).
- C'est un standard IEEE (1149.1) qui permet originellement de tester des circuits électroniques pour l'industrie des cartes PC.
- Le JTAG possède 3 entités :
 - le contrôleur TAP (Test Access Port). Machine à 16 états.
 - Un registre à décalage d'instructions.
 - Un registre à décalage de données.

INTRODUCTION

- Le JTAG forme une boucle série interconnectant les broches d 'E/S des circuits à surveiller/analyser (un point d 'entrée, un point de sortie).
- La boucle peut être active (écriture) et/ou passive (lecture). On récupère ainsi un flux de données série plus ou moins important. On va donc balayer périodiquement les composants JTAG (*boundary scan*).
- Chaque bit du registre à décalage de données du JTAG correspond à la valeur instantanée d 'une broche du circuit. On peut lire sa valeur courante ou lui affecter une nouvelle valeur (écriture).

INTRODUCTION

Each JTAG cell “sniffs” the state of the corresponding output bit of the IC

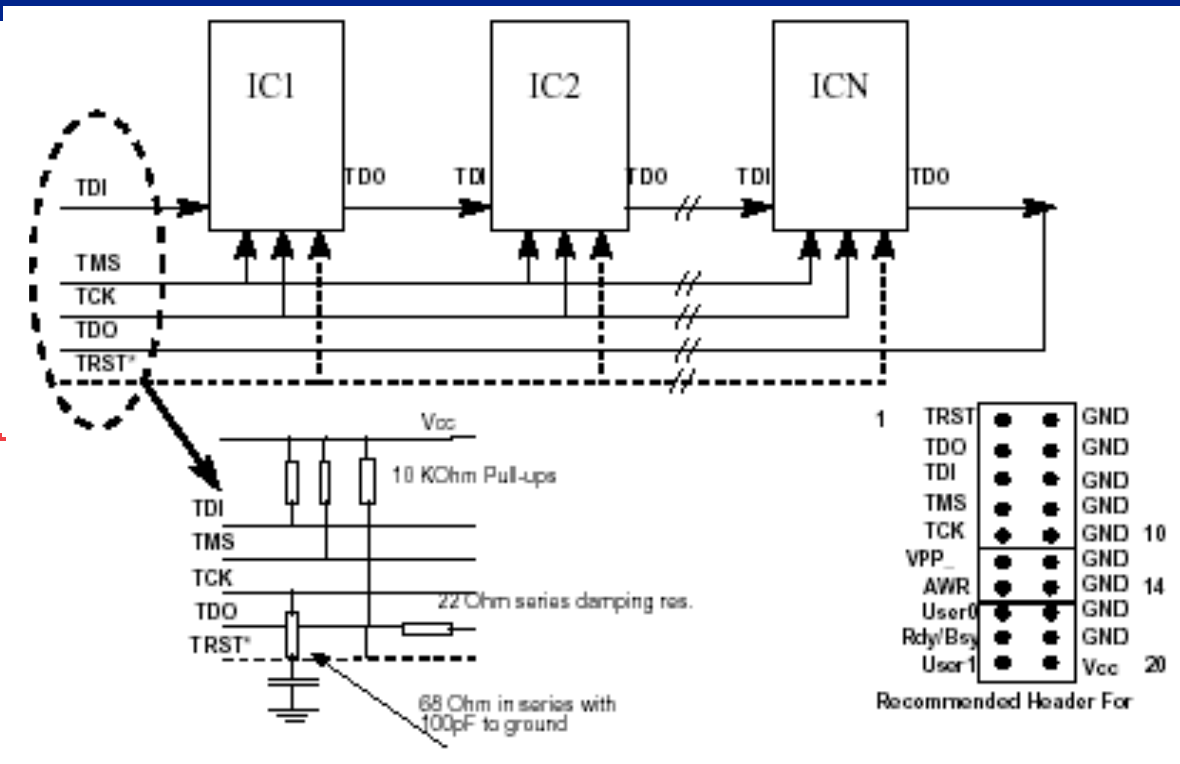
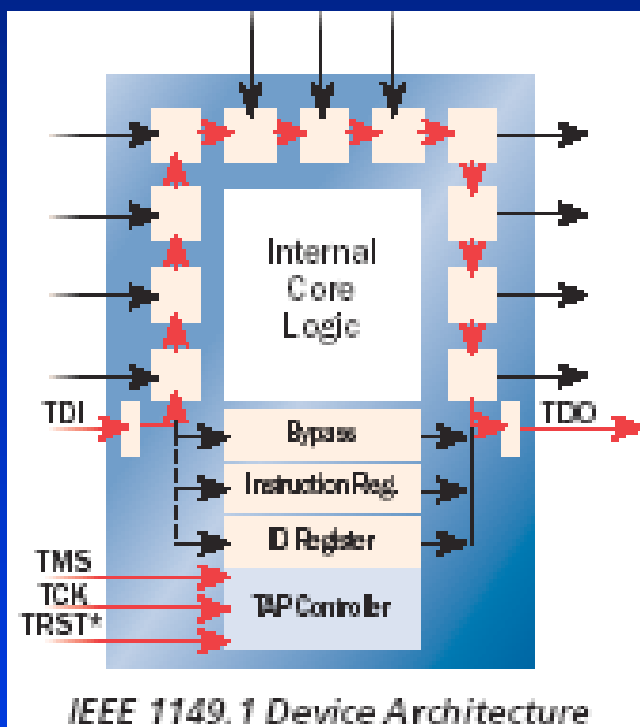


Bit stream forms one long shift-register

INTRODUCTION

- Le contrôleur TAP possède 4 broches :
 - TCK : horloge (Test ClOcK).
 - TMS : mode test de contrôle du TAP (Test Mode Select).
 - TDI : entrée donnée série (Test Data In).
 - TDO : sortie donnée série (Test Data Out).

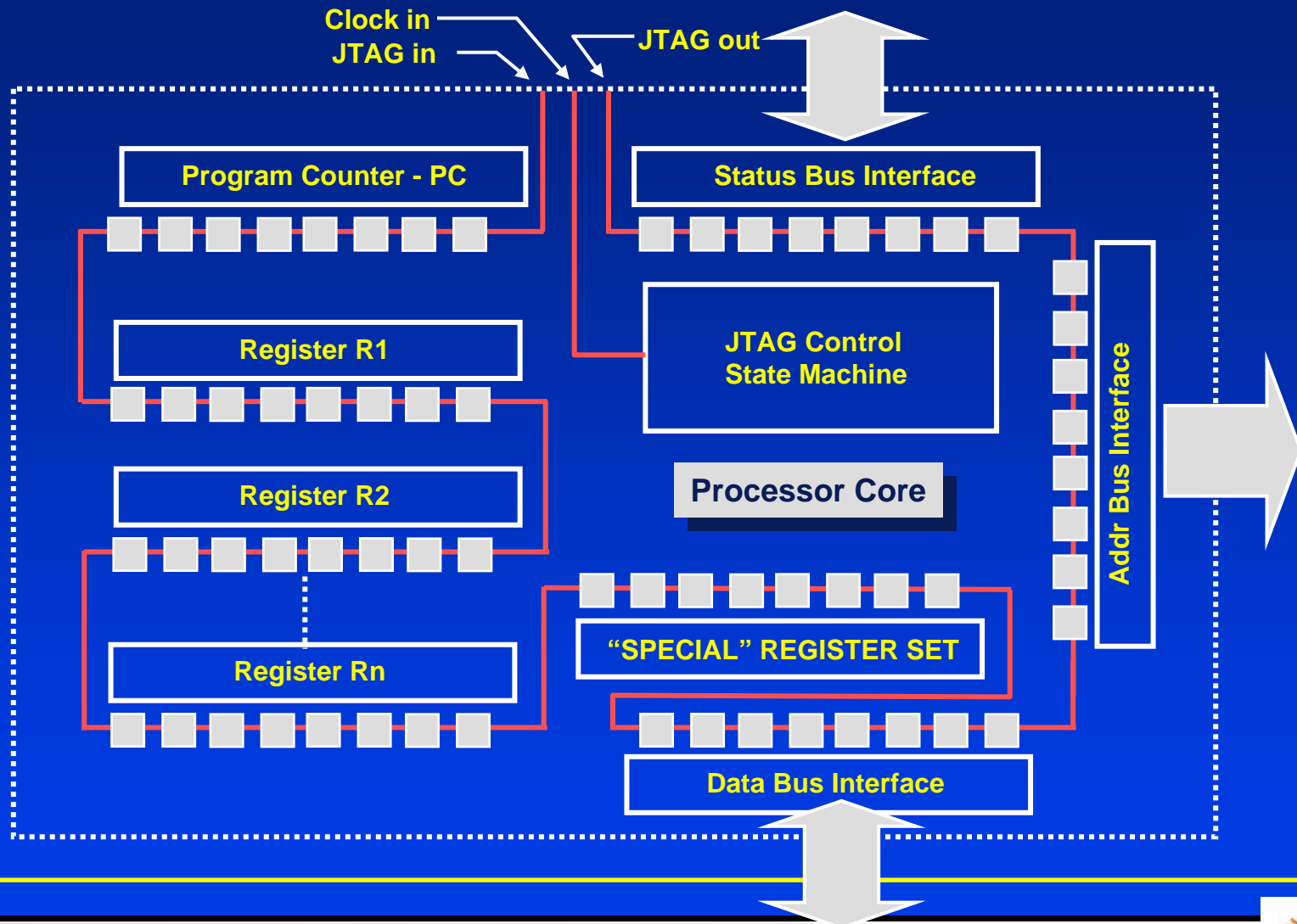
INTRODUCTION



DEBUG PAR LE JTAG

- Il est possible de mettre au point un système par le JTAG : la norme JTAG a été étendue par certains fondeurs pour cela.
- On lie tous les registres internes et les blocs fonctionnels importants par une boucle JTAG. On a donc un debugger simple sur silicium (*on chip*).
- On autorise la lecture comme l'écriture dans les registres.
- On peut rajouter des registres de debug spécifiques accessibles par le JTAG.

DEBUG PAR LE JTAG



AVANTAGES

- Le debugger JTAG est un debugger on chip (OCD On Chip Debugger). On peut mettre des points d'arrêt, faire du pas à pas, regarder la mémoire, changer une valeur en mémoire, télécharger du code...
- Le JTAG est un standard ouvert et non propriétaire comme le BDM.
- Il est adapté aux composants montés en surface.
- Il est bon marché.

AVANTAGES

- Il n ' utilise pas beaucoup de broches car il met en œuvre un protocole série.
- Il permet d ' observer des entrées et positionner des sorties sans utiliser la logique du système.
- Il permet de réduire les points de tests sur le système.
- Il ne marche qu ' avec un processeur le supportant. Si un processeur supporte le JTAG pour le debug, rajouter toujours le connecteur JTAG sur le PCB (on ne sait jamais) ou le connecteur BDM !

AVANTAGES

- Il peut être utilisé aussi pour la programmation de mémoire flash embarquée sur le système.
- Il supporte éventuellement des point d'arrêt matériels simples. Une broche spéciale suspend l'activité du CPU dans l'extension JTAG.

INCONVENIENTS

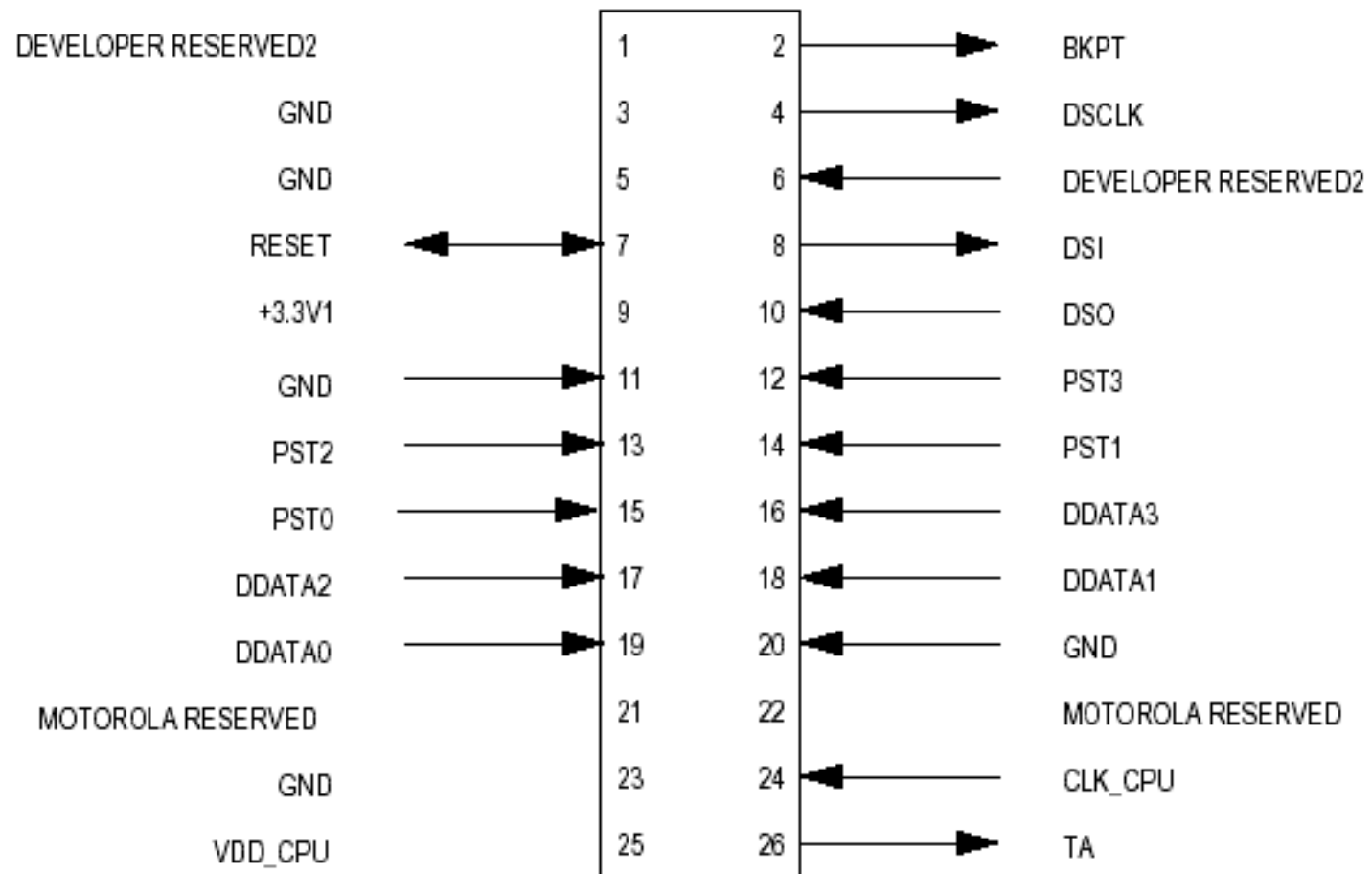
- Il peut être lent : chaque bit doit être décalé dans le registre à décalage. La boucle peut être grande (>10000 bits).
- Il n'a pas toujours de points d'arrêt hard.
- Il n'y a pas de points d'arrêt complexes comme avec l'émulateur ICE.

BDM

INTRODUCTION

- Le BDM (Background Debug Monitor) est un debugger on chip (OCD On Chip Debugger) propriétaire développé par Motorola pour ses processeurs CPU32 (683xx, ColdFire).
- Il utilise un connecteur 10 ou 26 broches sur le système cible.
- Un logiciel de debug sur l'hôte dialogue via la liaison BDM avec le debugger OCD en envoyant des commandes de debug de (16+1) bits émis en série.

INTRODUCTION



- NOTES: 1. Supplied by target
 2. Pins reserved for BDM developer use. Contact developer.

AVANTAGES

- Le BDM est bon marché. C 'est un OCD (On Chip Debugger).
- Le BDM a les avantages de mise au point d 'un monitor (on Chip) et offre en plus beaucoup des fonctionnalités d 'un émulateur ICE.
- Comme la logique BDM est à l 'intérieur du processeur, on peut voir toute l 'activité CPU et les accès aux caches.

INCONVENIENTS

- Il n'y a pas de points d'arrêt complexes.
- Il n'a pas toujours de points d'arrêt hard.

SIMULATEUR

SIMULATEUR

- Le simulateur (ISS Instruction Set Simulator) est un logiciel exécuté par le système hôte de développement pour simuler un processeur cible particulier.
- La principale difficulté est de simuler les périphériques réels du système cible.
- Les simulateurs sont plutôt réservés à de petits processeurs (microcontrôleur) :
 - MPLAB pour le simulateur PIC Microchip.

LE CHOIX DES OUTILS

LE CHOIX DES OUTILS

- Le bon sens doit avant guider l'ingénieur dans le choix des outils de debug à utiliser.
- On part généralement des outils les plus simples aux outils les plus sophistiqués :
 - L'oscilloscope : vérification des horloges, signaux critiques...
 - L'analyseur logique : reset, vérification des bus d'adresses et de données, accès mémoire.
 - L'émulateur ROM, ICE : couplage avec le logiciel. Debug au niveau source.
 - On peut ensuite embarquer un moniteur en ROM pour le lancement de l'application finale ou faire des tests en cours de production.

LE CHOIX DES OUTILS

Fonctionnalité	Emulateur ICE	BDM	ROM Moniteur	Analyseur logique	ROM Emulateur
Event triggers	Oui	Certains	Non	Oui	Non
Overlay RAM	Oui	Non	Non	Non	Oui
Points d'arrêt hard	Oui	Certains	Non	Non	Certains
Points d'arrêt complexes	Oui	Non	Non	Oui	Non
Time stamps	Oui	Non	Non	Oui	Non
Accès non intrusif	Oui	Oui	Non	Oui	Non
Coût	Très cher	Bon marché	Bon marché	Cher	Bon marché

LE CHOIX DES OUTILS

Fonctionnalité	Emulateur ICE	BDM	ROM Moniteur	Analyseur logique	ROM Emulateur
Debug au niveau source	Oui	Oui	Oui	Certains	Oui
Téléchargement code	Oui	Oui	Oui	Non	Oui
Pas à pas	Oui	Oui	Oui	Non	Oui
Points d'arrêt	Oui	Oui	Oui	Non	Oui
Display/modifs registres	Oui	Oui	Oui	Oui	Oui
Regarder variables	Oui	Oui	Oui	Oui	Oui
Traces TR	Oui	Certains	Non	Oui	Non

PARTIE 3 : LA MISE AU POINT D 'UN SYSTEME EMBARQUE

ENVIRONNEMENT DE DEVELOPPEMENT

INTRODUCTION

- Développer et mettre au point une application pour un système embarqué est un art difficile à maîtriser pour une application allant de la simple superboucle (boucle plus interruptions) jusqu'à un ensemble de processus coopératifs exécutés par un système d'exploitation embarqué.
- On a généralement un environnement de développement croisé avec :
 - Une machine hôte (host) pour le développement et la mise au point.
 - Un système cible ou target dans lequel on va télécharger l'application puis l'exécuter dans la phase de mise au point.
- La mise au point fera appel aux outils précédents : émulateurs ICE ou ROM, BDM, JTAG, moniteur.

ENVIRONNEMENT DE DEVELOPPEMENT

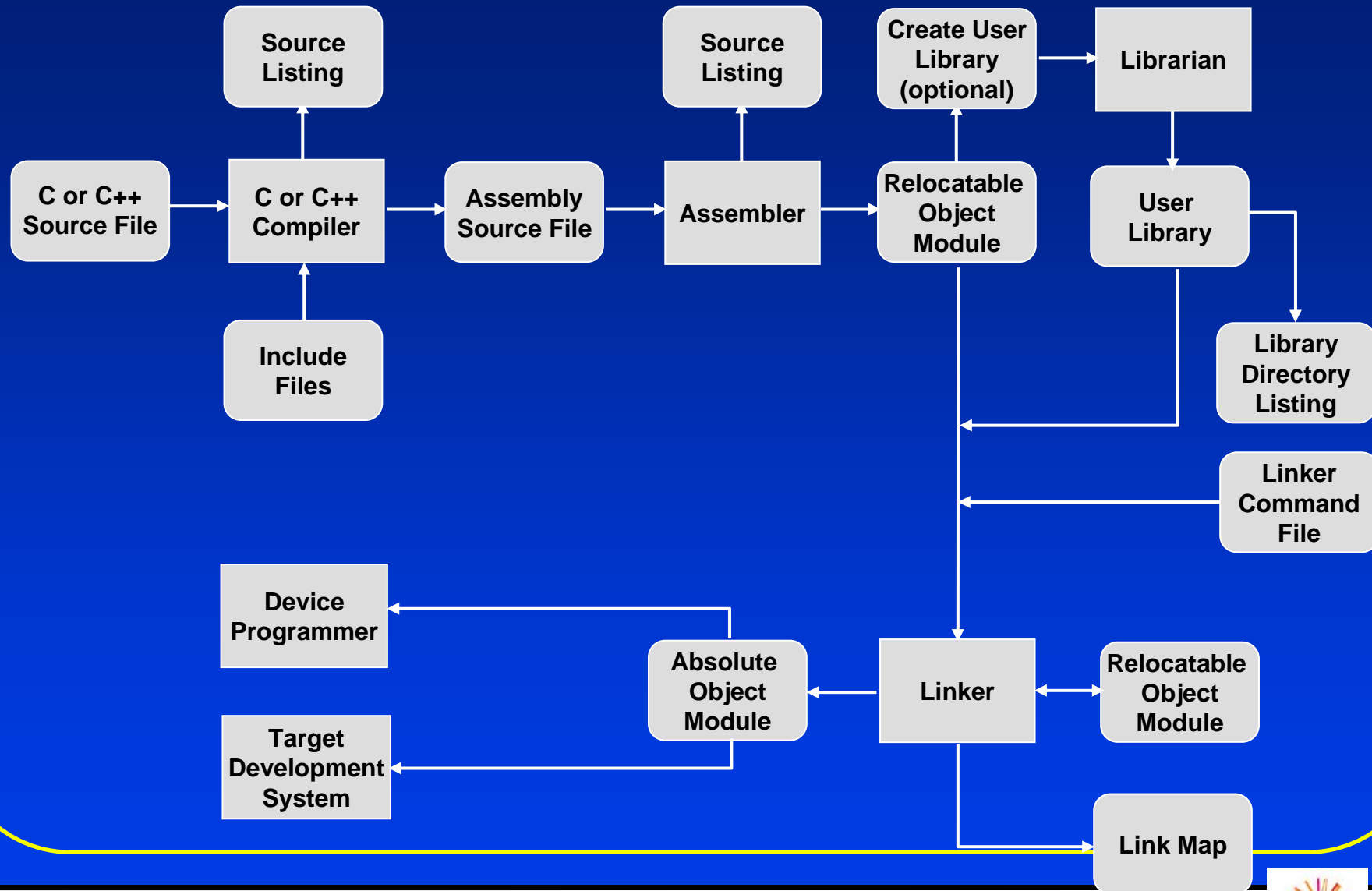
- On a un environnement de développement croisé :



ENVIRONNEMENT DE DEVELOPPEMENT

- On va développer et tester l'application sur une plateforme standard (PC) avec des outils logiciels conviviaux afin de faciliter le debug.
- On aura un compilateur croisé sur l'hôte pour avoir un code objet exécutable par le processeur de la cible.
- On aura un debugger croisé sur l'hôte pour mettre au point l'application exécutée par le processeur de la cible.

ENVIRONNEMENT DE DEVELOPPEMENT



ENVIRONNEMENT DE DEVELOPPEMENT

- Si l'on travaille en langage évolué (langage C), on teste au niveau du source en langage évolué (source level debugging).
- Le debugger croisé (au niveau source) sera couplé avec un équipement de debug (ICE, émulateur ROM, BDM, JTAG...).
- Il doit supporter le mode trace en Temps Réel pour capturer le traitement d'une interruption (ISR) sans ralentir le système.

MISE AU POINT LOGICIELLE D'UN SYSTEME EMBARQUE

INTRODUCTION

- Il ne suffit pas savoir programmer, il faut savoir bien programmer !
- Un système embarqué doit être robuste et son code bien écrit.
- Le langage de prédilection pour le développement logiciel est le langage C.
- Le langage C est un langage de haut niveau mais proche du matériel.
- Cette partie met le doigt sur certains points de la programmation en langage C auxquels il faudra faire très **attention**.

GENERALITES

- Se méfier des options d'optimisation du compilateur.
- Bien traiter les interruptions utilisées par le système en renseignant les vecteurs concernés dans la table des vecteurs du processeur. Mise en place de la routine d'interruption ISR (*Interrupt SubRoutine*).
- Renseigner toute la table des vecteurs même pour les interruptions non utilisées (ISR dummy...).

CLASSES DE STOCKAGE

- Pour un système embarqué, on est concerné par la façon dont le compilateur (C) va stocker les variables.
- On doit garder le contrôle du placement/mapping mémoire.
- La classe de stockage d'une variable peut être :
 - auto : auto int a;
 - Classe de stockage par défaut pour une fonction.
 - Stockage sur la pile.
 - Variable détruite (désallouée) en fin d'exécution de la fonction.

CLASSES DE STOCKAGE

- La classe de stockage d'une variable peut être :
 - register : register int a;
 - Stockage de la variable dans un registre du processeur.
 - Accès à la variable très rapide.
 - Limitation par le nombre de registres de données.
 - static : static int a;
 - Contraire de auto.
 - Dans le cas d'une telle variable de fonction, elle existe durant toute la durée d'existence de la fonction.
 - Si c'est une variable globale, elle existe durant toute la durée d'exécution du programme.

CLASSES DE STOCKAGE

- La classe de stockage d'une variable peut être :
 - extern : extern int a;
 - Variable définie dans un autre fichier source.
 - Variable importée.
 - Utilisé durant la phase d'édition de liens pour résoudre les références croisées.

MODIFICATEURS D 'ACCES

- Il existe 2 modificateurs d 'accès à une variable :
 - const : `const int a=10;`
 - La valeur de la variable ne peut pas être changée par le programme.
 - A placer en mémoire ROM.
 - volatile : `volatile char a;`
 - La valeur de la variable peut changer d 'elle-même en cours d 'exécution du programme.
 - Registre d 'E/S mappé en mémoire.
 - Zone mémoire servant de buffer par un périphérique externe (contrôleur DMA).

MODIFICATEUR D 'ACCES VOLATILE

- Le modificateur d 'accès volatile est à utiliser **obligatoirement** :
 - Pour l 'accès à des registres de périphériques mappés en mémoire.
 - Variables globales modifiées par une ISR.
 - Variables globales dans une application multi-tâche.
- Source : Introduction to the Volatile Keyword.N.Jones, Embedded Systems Programming.
<http://www.embedded.com/story/OEG20010615S0107>

VOLATILE : MAUVAISE PROGRAMMATION

- Exemple : scrutation active d'un registre d'état d'un périphériques 8 bits.
- Le registre d'état 8 bits est à l'adresse 0x1234.
- Scrutation active du registre d'état jusqu'à ce qu'il soit non nul.

VOLATILE : MAUVAISE PROGRAMMATION

```
unsigned char * ptr = (unsigned char *) 0x1234;
```

```
// Wait for register to become non-zero
```

```
while (*ptr == 0)
```

```
;
```

```
// Do something else
```

- Cela risque de ne pas marcher dès que le compilateur optimise le code. Le code généré pourrait être :

```
mov ptr, #0x1234
```

```
mov a, @ptr
```

```
loop bz loop
```

VOLATILE : MAUVAISE PROGRAMMATION

- Le compilateur C voit qu 'il a déjà la valeur courante de la variable.
- Il n 'y a aucun besoin d 'aller la relire puisque c 'est la toujours même valeur (la valeur d 'une case mémoire ne change pas d 'elle même !).
- Il génère alors une boucle infinie.
- On ne sort donc pas de le boucle !
- D 'où un BUG en run time !

VOLATILE : BONNE PROGRAMMATION

```
volatile unsigned char * ptr = (volatile unsigned char *)  
    0x1234;
```

- Le code assembleur généré est alors :

```
    mov     ptr, #0x1234  
loop    mov  a, @ptr  
        bz   loop
```

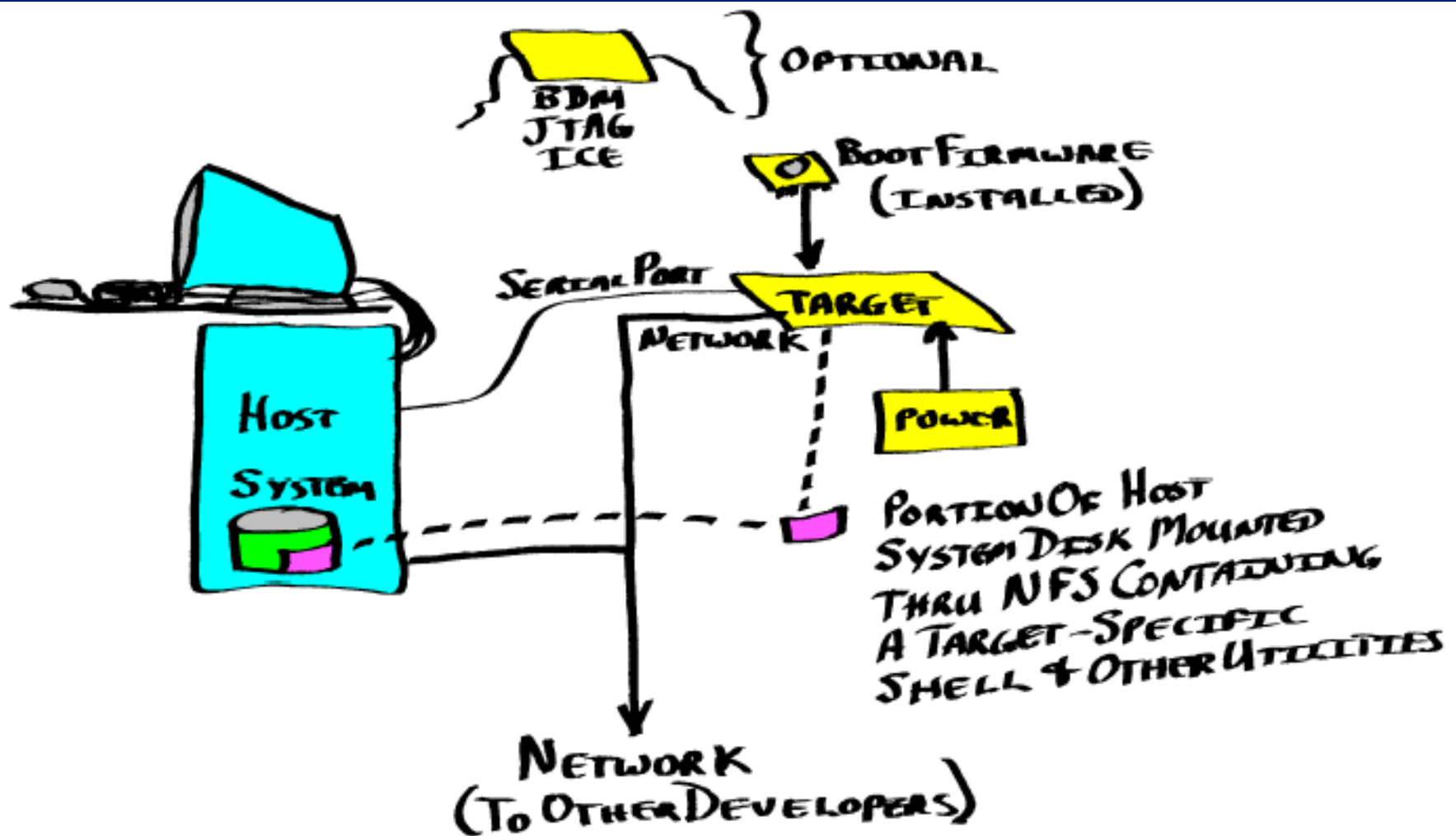
On a alors le comportement attendu !

PARTIE 4 :
EXEMPLE ILLUSTRÉ : MISE AU POINT
D'UN SYSTÈME LINUX EMBARQUÉ

LINUX EMBARQUE : LES ETAPES

- 1. Développement du bootloader.
- 2. Portage du noyau Linux.
- 3. Développement des drivers spécifiques.
- 4. Développement de l'application finale.
- 5. Intégration du tout.

L'ENVIRONNEMENT



LE SYSTEME HOTE OU HOST

- Utilisation d'un PC ou une station de travail comme host sous Linux en général (à défaut sous Windows avec Cygwin).
- Connexion par liaison série (ou par réseau Ethernet maintenant) avec le système embarqué cible ou target : minicom, hyperterminal
- Partage de fichiers avec la cible par montage NFS si la cible a une interface réseau Ethernet.
- Développement croisé sur l'hôte avec tous les composants logiciels nécessaires.

LE SYSTEME CIBLE OU TARGET

- Utilisation d'une carte CPU qui exécute Linux.
- On a besoin d'y intégrer différents composants logiciels :
 - une boot ROM contenant un *bootloader* chargeant le noyau Linux en mémoire et puis l'exécutant.
 - Un noyau Linux (compressé).
 - Une image disque d'un système de fichiers Linux contenant les programmes applicatifs, bibliothèques partagées, modules Linux, fichiers de script...

TELECHARGEMENT DANS LA CIBLE

- Utilisation d'un équipement particulier pour le téléchargement dans la cible en phase de développement (en final, tout sera « romé ») :
 - ICE, JTAG, BDM...
- On flashera en premier en mémoire FLASH le bootloader.
- Le bootloader pourra ensuite télécharger le noyau et l'image du système de fichiers par la liaison série ou par le réseau (TFTP) (et le mettre en mémoire FLASH au final...).
- Le système de fichiers (root File System) peut être éventuellement monté par NFS au lancement du noyau Linux de la cible...

ENVIRONNEMENT DE DEVELOPPEMENT CROISE

- Environnement de développement croisé sous Linux. On utilisera les outils GNU :
 - gcc, binutils.
 - ld, as, nm, obcopy, strip, ar...
- Ces outils permettent de créer une image binaire du noyau et du système de fichiers (root file System) sur l'hôte.
- On pourra utiliser un atelier de génie logiciel (IDE : Integrated Design Environment) : Kdevelop, Metrowerks Code Warrior...

ENVIRONNEMENT DE DEVELOPPEMENT CROISE

- Utilisation du debugger GNU ou GDB.
- Debug par la liaison série ou par le réseau. On a sur la cible un programme serveur gdbserver qui attend les ordres d'une application cliente exécutée par l'hôte. On bénéficie alors d'une interface graphique conviviale sur l'hôte qui permet de debugger au niveau source :
 - utilisation d'un « front end » à gdb comme DDD (Data Display Debugger).
- On pourra aussi mettre à profit les outils de base Linux sur l'hôte pour la gestion du projet :
 - make, CVS (Current Version System)...

DEVELOPPEMENT VS PRODUCTION

- En phase de développement, on a à disposition des outils :
 - Qui permettent un téléchargement rapide.
 - Qui permettent de télécharger le noyau Linux.
 - Qui permettent de télécharger l'application par NFS.
 - Qui permettent de développer, de tester sur l'hôte puis de porter sur la cible.
- En phase de production, le bootloader, le noyau et son root File System sont romés en mémoire FLASH :
 - Taille totale ?
 - Comment mettre à jour (correction de bugs...) ?

QUOI DEVELOPPER ?

- On doit développer :
 - Le système électronique (procédure classique).
 - Le bootloader.
 - Le portage du noyau.
 - Les pilotes de périphériques.
 - L 'application.
 - L 'intégration du tout.

DEVELOPPEMENT DU BOOTLOADER

- Développement du bootloader :
 - En utilisant les sources éventuels fournis avec la carte si on l'a achetée.
 - En utilisant un bootloader open source : U-boot, RedBoot, Colilo...
 - Se l'écrire soi-même.
- Le bootloader est en mémoire FLASH. Il prend en charge l'initialisation de base du système, les autotests, la décompression de l'image+root FS stockés en mémoire FLASH ou téléchargé (liaison série ou réseau).
- Il peut avoir des fonctions de debugger (moniteur).

PORTAGE DU NOYAU

- Il est important de comprendre comment marche le noyau et notamment la procédure de boot.
- Il faut choisir le portage correspondant au type de processeur de la cible.
- Il faut modifier éventuellement les sources du noyau (fichiers ASM) pour coller au hardware de la carte cible.

DEVELOPPEMENT DES DRIVERS

- Il est souhaitable de choisir des composants qui sont supportés par Linux lors de la phase de conception du système.
- Lors de la phase de configuration du noyau, on choisira les drivers appropriés. On pourra éventuellement les modifier pour coller au mapping mémoire de la cible.
- Si l'on doit écrire un nouveau driver pour un matériel spécifique, il est préférable de partir d'un driver existant similaire.
- Il faudra faire le choix entre développer un driver statique lié au noyau ou un driver dynamique (module Linux) chargé en mémoire par le noyau à la demande.

DEVELOPPEMENT DE L 'APPLICATION

- Il est important de voir s 'il n 'existe pas déjà une application existante collant à son besoin. Dans le cas éventuel, la porter sur sa cible. Dans le meilleur des cas, cela se traduira par une simple recompilation croisée.
- Choisir de préférence des applications open source.
- Considérer le développement d 'une interface graphique (GUI Graphical User Interface) si besoin :
 - système de type Xwindow.
 - Frame buffer.
 - Microwindows, Qt/embedded...

INTEGRATION SYSTEME

- Il faudra veiller à :
 - Evaluer la taille mémoire consommée (RAM, FLASH)
 - Prendre en compte la faible empreinte mémoire :
 - Chargement du noyau en RAM ou XIP (eXecute In Place).
 - Root File System en RAM ou FLASH.
 - Type du système de fichiers : ext3, JFFS2 pour un système embarqué.
 - Bibliothèques : statiques ou dynamiques.
 - Bibliothèques libc light (μ Clibc...).

CHAPITRE 4 : LINUX EMBARQUE

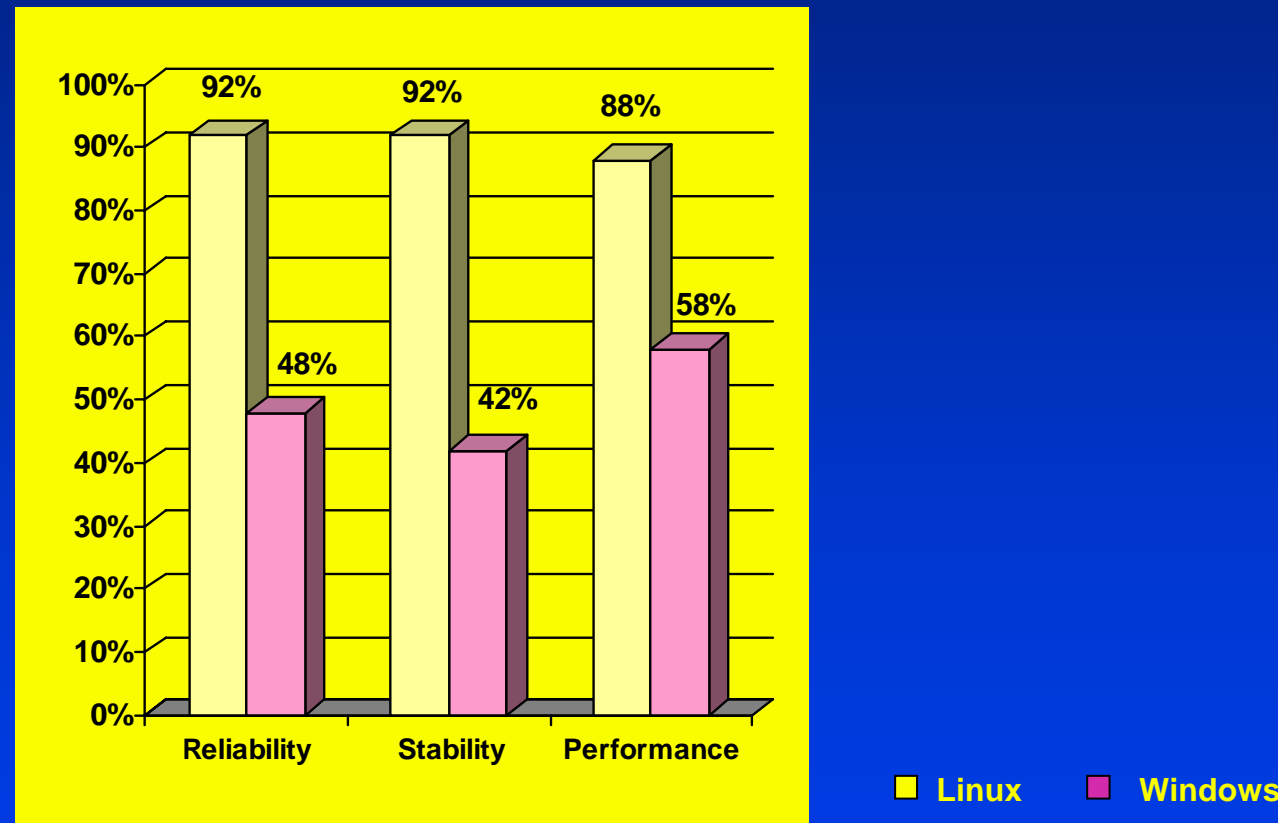
PARTIE 1 : LE BESOIN D'EMBARQUER LINUX

L 'OPPORTUNITE DE LINUX SUR MARCHE DE L 'EMBARQUE

- Beaucoup sont passés d 'un OS propriétaire (Microsoft) à Linux pour l 'embarqué malgré encore quelques réticences archaïques :
 - Quelque chose de gratuit est de la camelote (voir le prix plancher psychologique d 'un produit au supermarché).

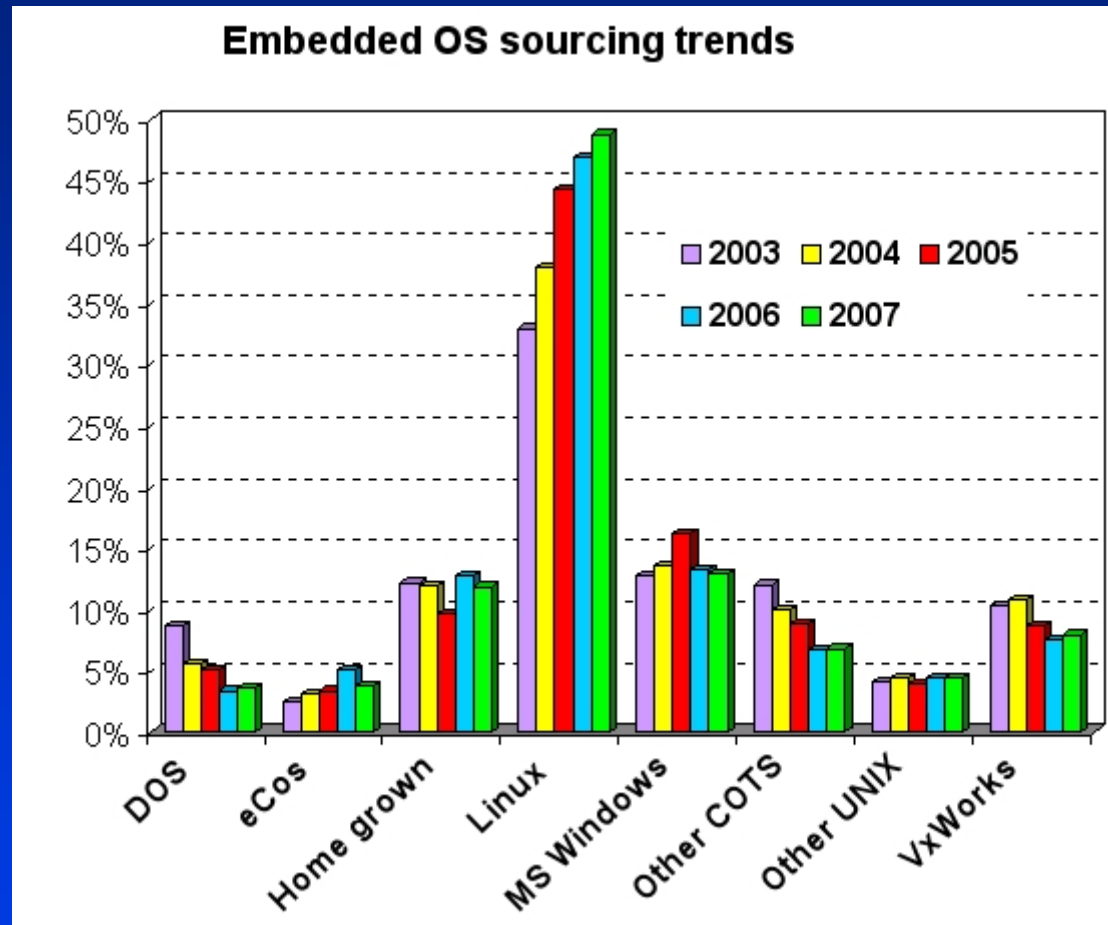
L'OPPORTUNITÉ DE LINUX SUR MARCHÉ DE L'EMBARQUE

- On retiendra les comparaisons suivantes (d'après www.survey.com) :

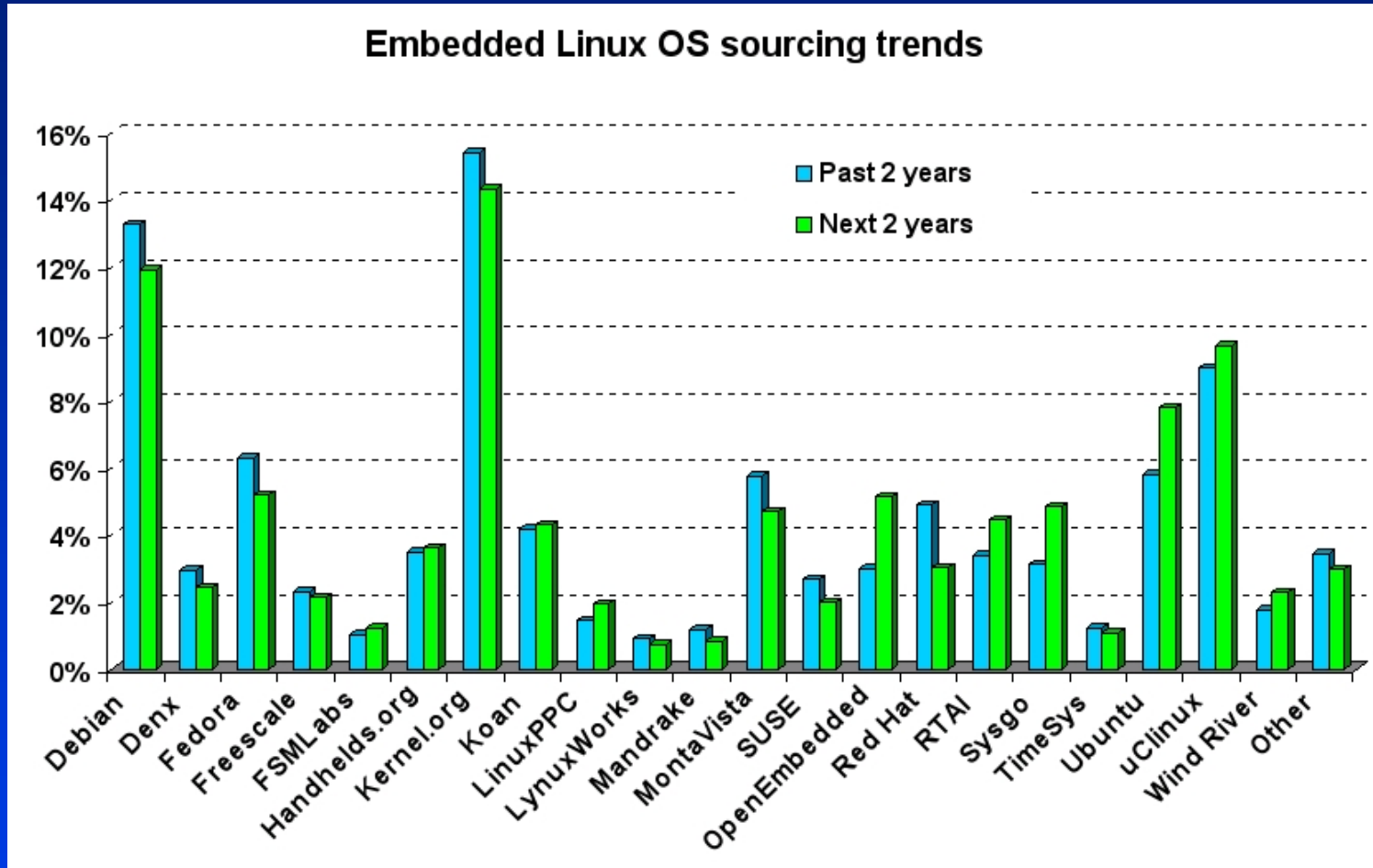


LE MARCHE DE L'EMBARQUE

- Panorama du marché de l'embarqué :

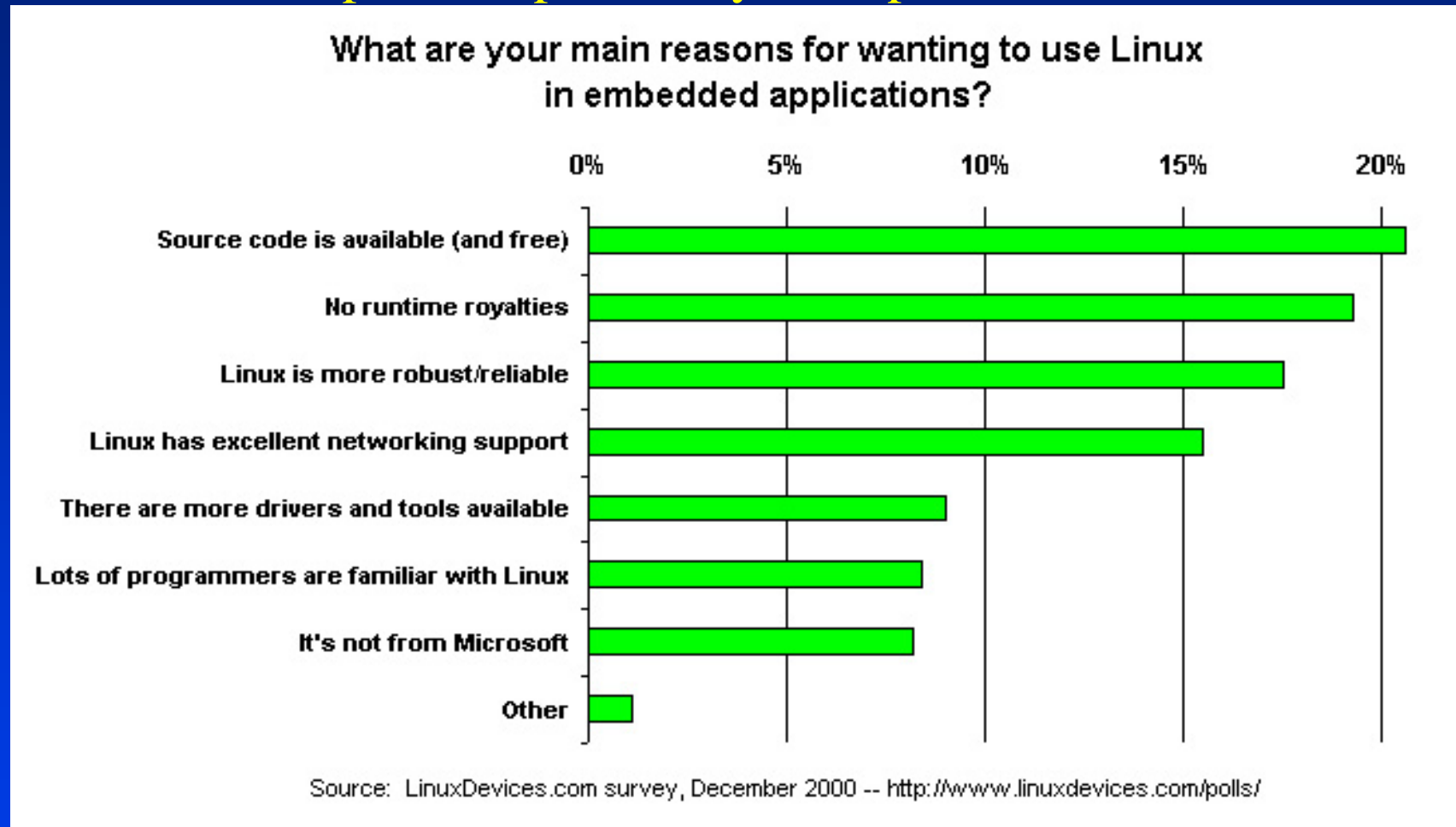


LE MARCHE DE L'EMBARQUE

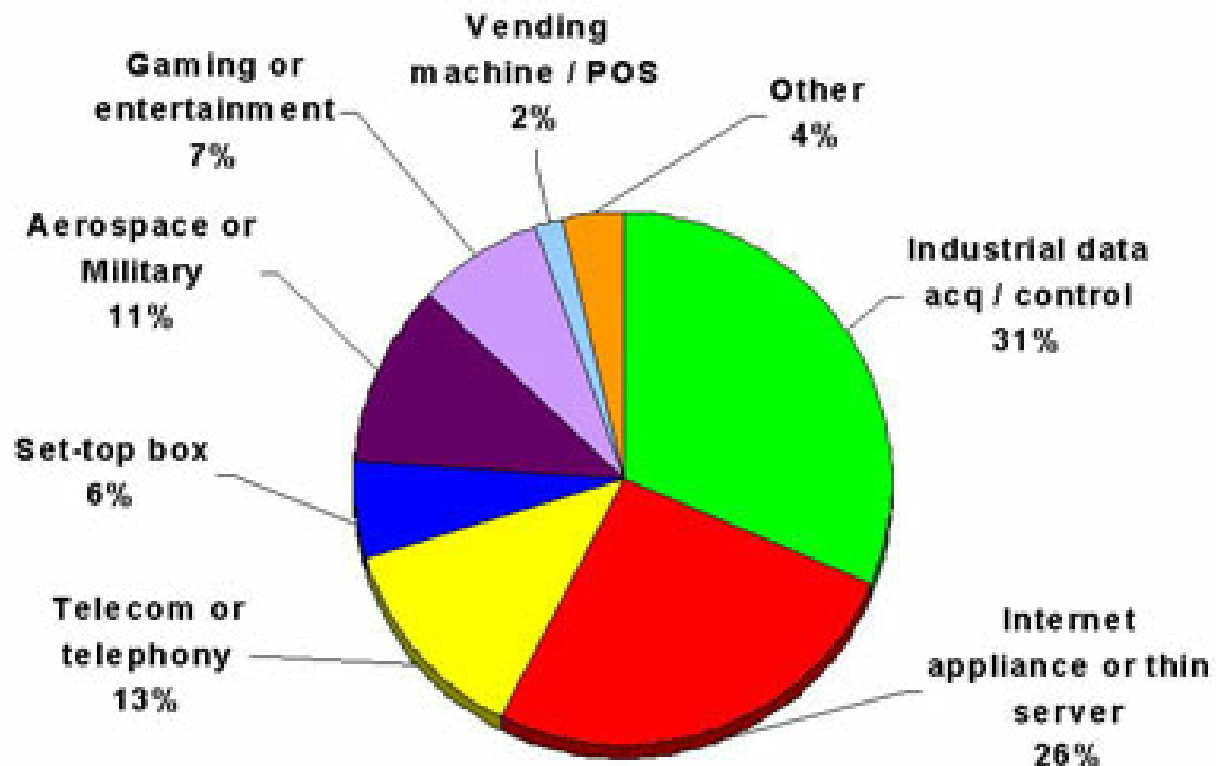


POINTS FORTS CITES

- Code source disponible, pas de royalties pour Linux :



APPLICATIONS VISEES PAR LINUX



(Copyright ©2001, CNET Networks, Inc.)

QU 'EST-CE QUE LINUX ?

- Linux est un système d 'exploitation libre de type UNIX lancé par le finlandais Linus Torvalds en 1991 avec l 'assistance de milliers de développeurs dans le monde pour son évolution.
- Son succès tient au fait qu 'il est développé sous licence GPL (*General Public License*), ce qui signifie que le code source Linux est disponible à tout le monde et gratuit.
- Son emblème est un pingouin : le *tux*.

QU 'EST-CE QUE LINUX ?

- Linux correspond au cœur du système d 'exploitation : le noyau.
- Linux tourne originellement sur plateforme i386 et supérieure avec 8 Mo de RAM.
- **IL FAUT DONC UN PROCESSEUR 32 BITS AVEC MMU (OU A DEFAUT 32 BITS SANS MMU AVEC μ Clinux)**

QU'EST-CE QUE LINUX ?



- Linux est complété des outils/logiciels GNU (*Gnu is Not UNIX*).
- Linux est disponible sous forme de distributions : Debian, RedHat, Mandrake, SuSE, Slackware...
- Linux est utilisé avec une interface graphique comparable à Microsoft Windows : Gnome, KDE

POURQUOI UTILISER LINUX ?

- Linux est open source :
 - Le code source est disponible au public.
 - Le code source inclut :
 - Le noyau Linux.
 - Les pilotes de périphériques (*drivers*).
 - Un ensemble de petits utilitaires (MAKEDEV...).
- On peut ainsi voir directement à travers les fichiers sources ce que fait le noyau Linux voire modifier son comportement au besoin. On n'a donc pas une boîte noire (avec comme seul interlocuteur une hot-line !).

POURQUOI UTILISER LINUX ?

- Linux est fiable :
 - Grâce à une gestion mémoire optimisée, Linux peut tourner sur une machine des années sans plantage et sans « écran bleu de la mort ».
- Linux est extensible :
 - Une application Linux écrite pour une plateforme PC peut être facilement portée sur une plateforme Linux embarquée.
 - Cette même application peut être aussi facilement portée sur un cluster Linux (grappe d'ordinateurs coopératifs).

POURQUOI UTILISER LINUX ?

- Linux est sécurisé :
 - Linux est recommandé par le NSA américain.
 - Linux est conçu pour que les processus ne puissent pas lire en mémoire code et données sans provoquer une violation des règles de sécurité du système (*segmentation violation*). Cela permet de confiner les programmes malicieux.
 - Sécurisation du système de fichiers avec des droits d'accès.
 - Sécurisation d'accès physique à la plateforme.
 - Sécurisation de l'accès réseau.

POURQUOI UTILISER LINUX ?

- Linux supporte la plus large palette de protocoles réseau testés et éprouvés (indispensable pour la connectivité IP dans l'embarqué) :
 - TCP/IP networking.
 - Routing/Firewalling.
 - Web Server.
 - FTP Server.
 - Telnet Server.
 - SMB.
 - NFS.
 - protocoles WAN : X.25, AX.25, HDLC, ATM.
 - ...

POURQUOI UTILISER LINUX ?

- Linux possède un support efficace à travers la communauté de développeurs.
- On trouve toujours une application Linux correspondant à son besoin (ou très proche).
- On capitalise son expérience UNIX en travaillant sous Linux car Linux est UNIX like d'où des coûts de formation réduits.

POURQUOI UTILISER LINUX ?

- Les coûts de mise en œuvre de Linux sont réduits :
 - Toutes les distributions Linux sont disponibles gratuitement au téléchargement par Internet.
 - On peut acheter une distribution (< 150 euros) avec la documentation papier et un service support de 30 jours généralement.
 - Les outils de développement (compilateurs, IDE...) sont disponibles à faible coût ou gratuits (GNU).

POURQUOI UTILISER LINUX ?

- Linux est **sans royalties** à payer pour chaque produit vendu à base de Linux.
- Ce point est une (r)évolution dans le domaine de l 'embarqué où les outils (OS, IDE...) sont chers et où l 'on paye en plus des royalties non négligeables sur chaque produit conçu avec.

LINUX ET LE LOGICIEL LIBRE

- Linux est un logiciel libre : cela donne le pouvoir aux utilisateurs d'utiliser ce logiciel comme ils l'entendent :
 - **“Free software is a matter of liberty, not price ...‘free’ as in ‘free speech,’ not as in ‘free beer’...”**. Free Software Foundation
- Le développement n'est pas contrôlé par un petit groupe de développeurs donc pas de despotisme possible.
- Il est possible de gagner de l'argent avec le logiciel libre (formation, assistance...).

L 'OPEN SOURCE

- L '*open source* : accès aux sources du logiciel.
- L 'open source permet :
 - Une interopérabilité entre applications et les différentes plateformes.
 - La formation par analyse des sources.
 - L 'accès aux sources permet d 'optimiser des parties de code pour des performances accrues.
 - Les idées et algorithmes deviennent des standards et sont disponibles à tous sans brevet.
 - Des distributeurs développent et vendent leurs fonctionnalités au dessus de logiciels open source.
 - Le terme *open source* est plus vendeur que logiciel libre.

LOGICIEL LIBRE : DROITS

- Définition technique (*free software*) d'après la FSF (*Free Software Foundation*) :

Users have the freedom to :

- (1) run the software, for any purpose;
- (2) study how the program works and adapt it to their needs;
- (3) redistribute copies;
- (4) improve the program and release improvements to the public

Access to source code is necessary for (2) and (4) so “Free” can include “Open Source”

LOGICIEL LIBRE : DROITS

- En conséquence, le logiciel libre peut être modifié, utilisé et même vendu.
- Vendre un logiciel libre correspond à ajouter un service, un bonus :
 - Outils d'installation, de packaging de logiciels.
 - Aide, support, formation.
 - Adaptation de logiciel à un besoin spécifique.
 - Driver d'un matériel sous forme d'un module Linux (fourniture du fichier objet .o).
- Des améliorations d'un logiciel libre peuvent être proposées par tous sous forme d'une nouvelle release.

LOGICIEL LIBRE : OBLIGATIONS

- Mise à disposition du code source.
- Les modifications apportées au programme doivent être clairement indiquées et datées (Changelog).
- Un programme sous GPL reste un programme sous GPL.

LOGICIEL LIBRE : OBLIGATIONS

Pour distribuer un programme sous GPL :

- Transmettre tous les droits que vous possédez.
- S'assurer que les destinataires reçoivent le code source ou peuvent se le procurer.
- Leur remettre la licence GPL afin qu'eux aussi connaissent leurs droits.

LICENCE OPEN SOURCE

- Une licence logicielle précise ce que l'utilisateur peut faire avec un logiciel et son code
- Une licence traditionnelle (commerciale) précise strictement l'utilisation du logiciel acheté.
- Une licence open source indique comment le code peut être utilisé, réutilisé et redistribué. La licence généralement mise en œuvre est la licence GPL.

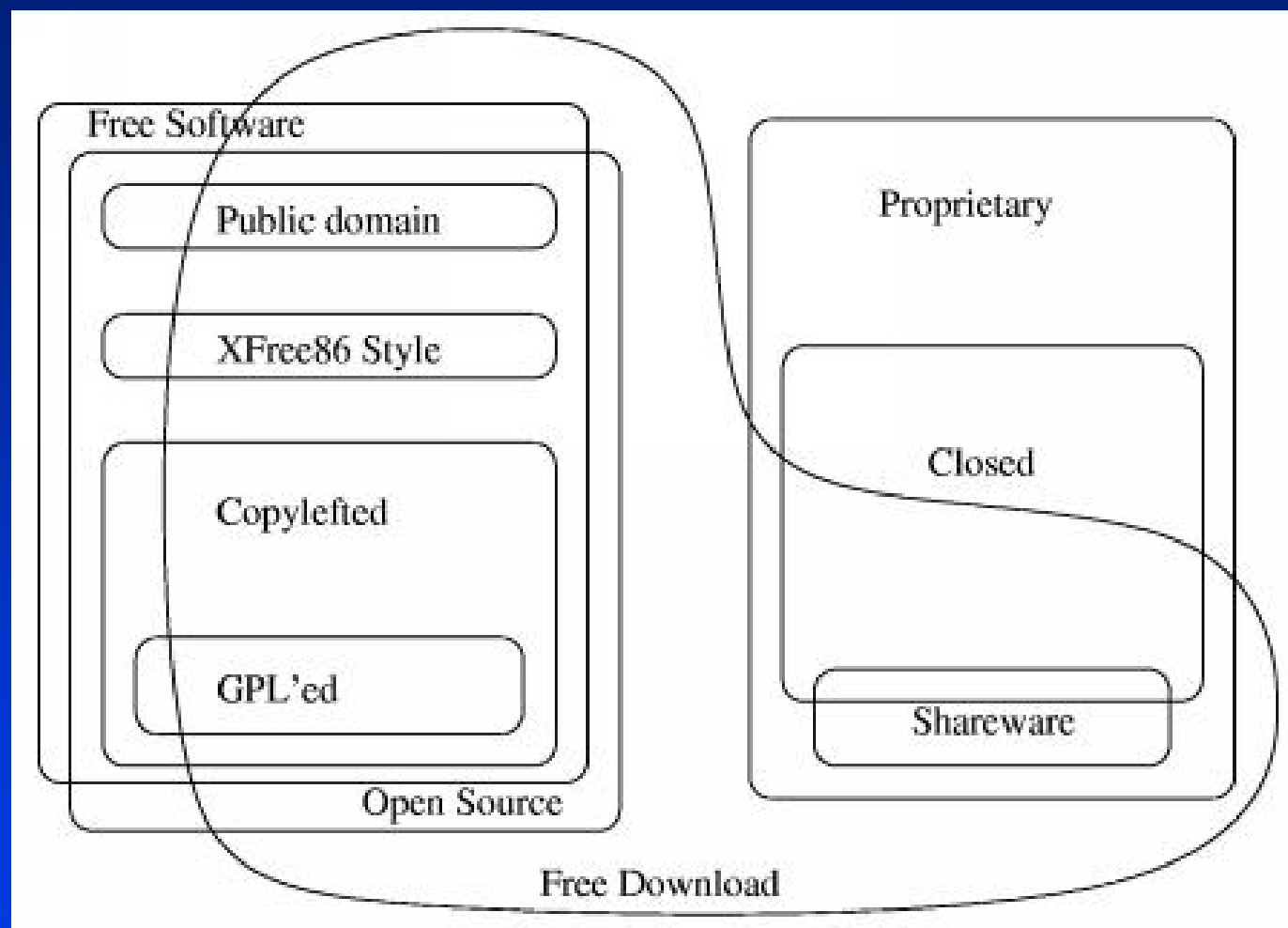
LICENCE OPEN SOURCE GPL

- La licence GPL est le modèle de distribution idéal d'un logiciel proposé par la FSF et le projet GNU.
- Les développeurs peuvent choisir de licencier leur logiciel sous licence GPL. Cela exige que les utilisateurs maintiennent le code source originel et indiquent clairement les changements opérés avant toute redistribution.
- Le code source est disponible, les utilisateurs peuvent le modifier, le compiler comme ils veulent.

AUTRES LICENCES

- Il existe d'autres types de licences :
 - MIT.
 - BSD.
 - X11, XFree86.
 - Netscape.
 - W3C.

AUTRES LICENCES



LINUX ET LA PORTABILITE

- Linux (et ses applications) est fortement portable.
- Une même application peut être utilisée (portée) sur :
 - Un nombre important de processeurs : x86, Alpha, ARM, StrongARM, MIPS, PowerPC, SPARC, m68k...
 - Un nombre important de plateformes ou BSP (*Board support Package*).
 - Un nombre important d'interfaces physiques avec le driver adéquat.
- Linux est donc capable d'exécuter la même application du PDA à l'ordinateur de bureau.
- Linux est un système d'exploitation de choix pour les systèmes embarqués. On parle de Linux embarqué.

LINUX EMBARQUE

- Linux embarqué est une adaptation du noyau Linux à un système embarqué. Suivant les capacités du système, on ne retrouve qu'une partie des fonctionnalités du noyau :
 - Moins de services disponibles.
 - Moins de mémoire requise (< 8 Mo).
 - Boot depuis une mémoire ROM.
 - Pas de clavier ou de souris requis.
 - Logiciels spéciaux pour piloter les périphériques du système (écran LCD, flash disk, Disk On Chip DOC, touch screen...).

LINUX EMBARQUE

- Une version de Linux embarqué peut être spécialement configurée pour coller à une plateforme ou application précise :
 - Linux embarqué pour routeur IP.
 - Linux embarqué sur PDA.
 - Linux embarqué pour microcontrôleur sans MMU.
 - Linux embarqué sur processeur 80286 et inférieur.
 - ...

OUTILS POUR LINUX EMBARQUE

- On utilise pour le développement sous Linux embarqué les outils traditionnels GNU :
 - (cross) compilateurs C/C++. C est préférable pour limiter la taille des exécutables.
 - IDE.
 - GDB.
 - Simulateur.

OUTILS POUR LINUX EMBARQUE

- On utilise pour le développement sous Linux embarqué un PC de développement sous Linux (l'hôte) avec une chaîne de compilation croisée en fonction du processeur embarqué sur le système (la cible).
- L'exécutable ainsi produit est téléchargé dans la cible pour pouvoir y être testé. On utilisera alors GDB pour déboguer l'application par le réseau que l'on pourra coupler avec une interface graphique de type DDD.
- Un montage NFS depuis la cible d'un répertoire du PC hôte permet de simplifier la phase de téléchargement.

OUTILS POUR LINUX EMBARQUE

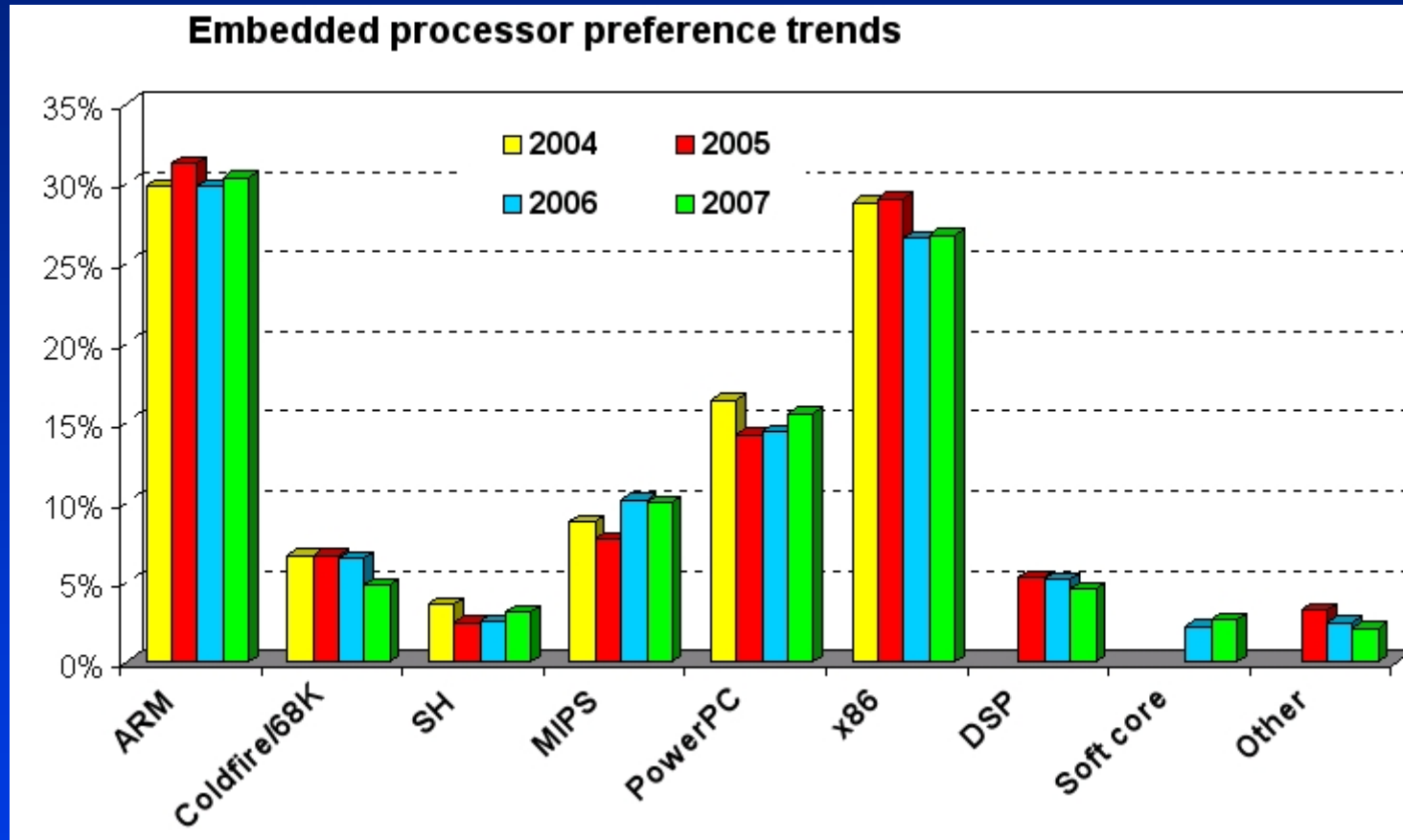
- Il existe des simulateurs tournant sur le PC hôte pour simuler la cible :
 - Simulateur pour émuler une grande marque de pocket PC.
 - Qemu.
 - UML (*User Mode Linux*).
- Cela permet de créer une machine virtuelle tournant un Linux embarqué correspondant à la cible et à son type de processeur.

LE CHOIX D 'UN PROCESSEUR POUR L 'EMBARQUE

Besoin	Miniature	Petit	Moyen	Haut de gamme	PC embarqué	Embarqué haute disponibilité
Taille RAM	<0,1 Mo	0,1-4 Mo	2-8 Mo	8-32 Mo	16-64 Mo	> x Mo
Taille ROM/FLASH	0,1-0,5 Mo	0,5-2 Mo	2-4 Mo FLASH	4-16 Mo FLASH	xx Mo	Go-To
Processeurs	DragonBall 68K Mcore ColdFire ARM		MIPS Hitachi SH x86 PowerPC		Pentium PowerPC	
Caractéristiques matérielles	MMU optionnelle		Ardoise Internet Carte unité centrale System on Chip (SoC)		CompactPCI	
Exemples d'applications	Caméra numérique PDA Téléphone		Routeur Décodeur Stockage en réseau Imprimante en réseau		Commutateur téléphonique Routeur haute performance Serveur central	

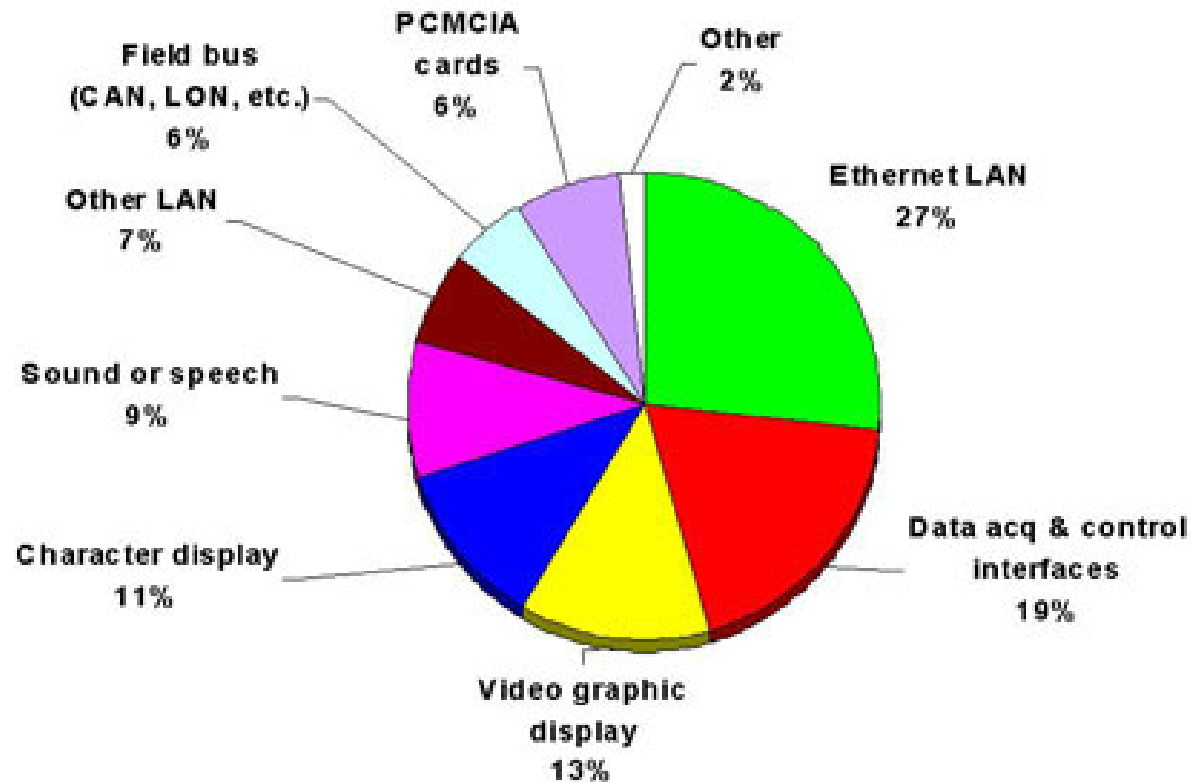
- Choix suivant puissance de calcul, taille mémoire...

CHOIX DU PROCESSEUR POUR LINUX EMBARQUE



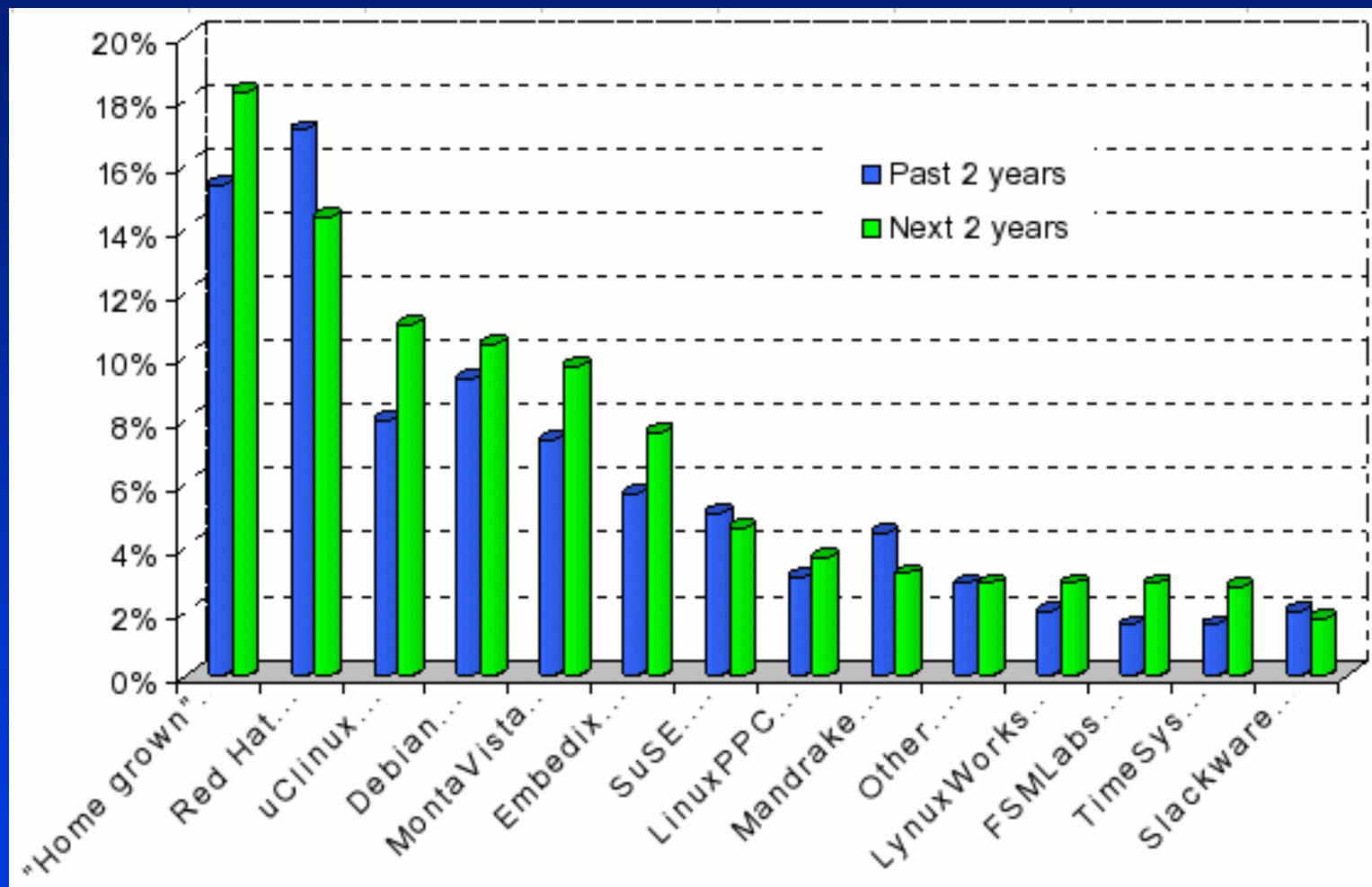
PERIPHERIQUES POUR LINUX EMBARQUE

What peripherals will be in the end system?



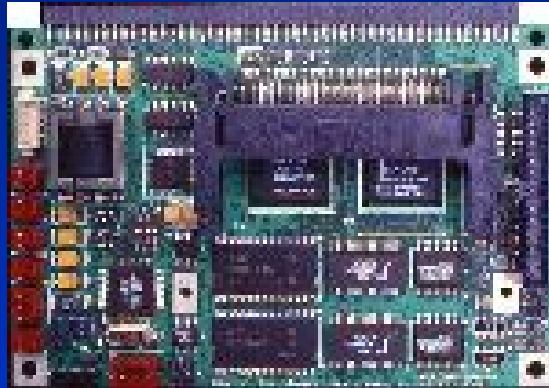
(Copyright ©2001, CNET Networks, Inc.)

CHOIX D'UN LINUX EMBARQUE



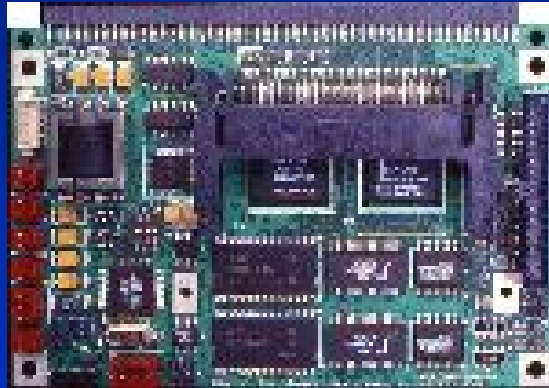
- Enquête linuxdevices.com juin 2003

CARTES POUR LINUX EMBARQUE



- Little Board (5.75 x 8.0 in.) -- complete systems on a single compact board, expandable with plug-on function modules
- ISA "slot boards" (full-length, 13.8 x 4.8 in.; half-length, 7.1 x 4.8 in.) -- IBM PC plug-in cards which could function as standalone SBCs backplanes)
- PC/104 modules (3.6 x 3.8 in.) -- compact, rugged, self-stacking modules featuring a reliable pin-and-socket board-to-board expansion bus

CARTES POUR LINUX EMBARQUE



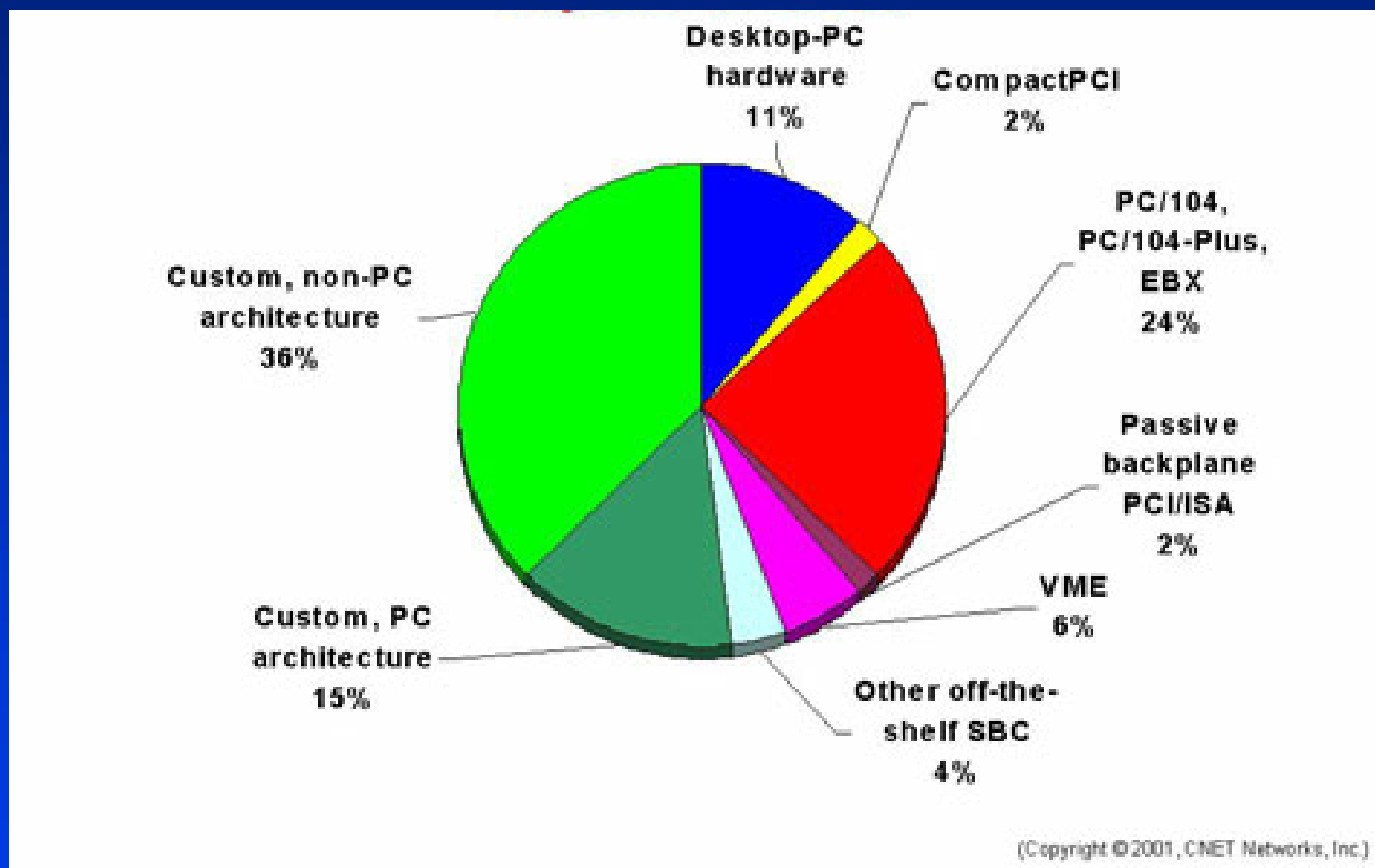
- Bus PCI en plus :

PC/104-Plus -- PCI added to PC/104

EBX -- PC/104-Plus added to Little Board

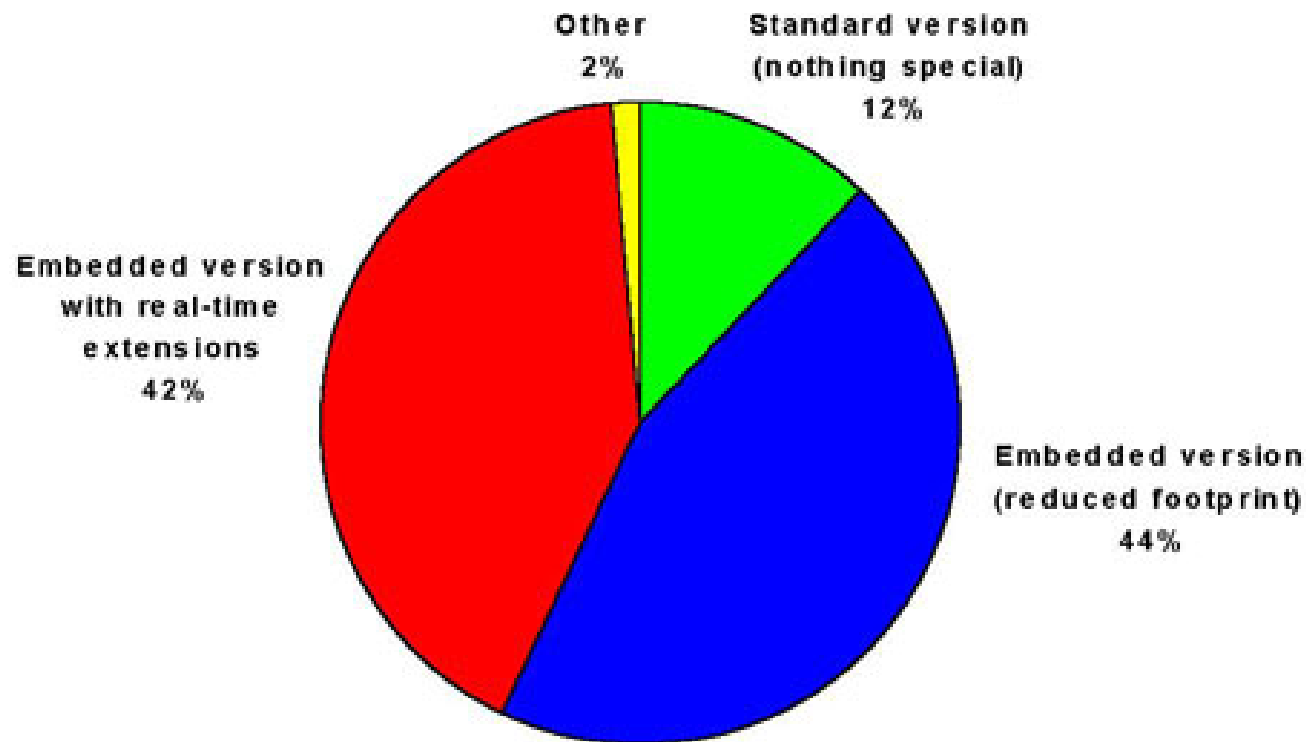
- Cartes au format industriel VME, VXI, PXI...

FORMAT DES CARTES CHOISI POUR LINUX EMBARQUE



CHOIX D'UNE VERSION LINUX EMBARQUE

What type of Linux OS will you need?



(Copyright ©2001, CNET Networks, Inc.)

POINTS FAIBLES DE LINUX EMBARQUE

- Les drivers Linux pour un périphérique donné ne sont pas toujours disponibles.
- Le manque de standards : window manager, GUI, extensions Temps Réel...
- Le manque d'une cohérence marketing.
- Le manque d'outils de qualification d'un système sous Linux (tests de conformité de l'API POSIX pour le Temps Réel ?).
- Le modèle de la licence GPL mal compris (droits et surtout obligations).

PARTIE 2 :

LES OFFRES LINUX EMBARQUE

LES OFFRES LINUX EMBARQUE

- Les offres de version de Linux embarqué (et Temps Réel) peuvent être rangées dans l'une des 3 catégories suivantes :
 - Les distributions Linux classiques : RedHat , Mandrake, Caldera, Debian, Slackware, Suse...
- Suivant la quantité de mémoire “disque” du système embarqué, il est possible d'édulcorer une distribution classique (<100-150 Mo). Cela tient dans une mémoire Compact Flash (512 Mo...).
- Le projet LFS (Linux From Scratch) explique comment construire son Linux pas à pas depuis rien suivant ses besoins :
<http://www.linuxfromscratch.org/>

LES OFFRES LINUX EMBARQUE

- Les offres de version de Linux embarqué (et Temps Réel) peuvent être rangées dans l'une des 3 catégories suivantes :
 - Les distributions Linux embarqué commerciales :
 - non TR : Montavista, Lineo-Metrowerks-Motorola/Creation Suite for Linux, LynuxWorks/BlueCat
 - TR : WindRiver-Intel/RTLinux, Montavista, LynuxWorks/BlueCat RT, TimeSys/Linux RTOS, Lineo-Metrowerks-Motorola/Creation Suite for Linux

LES OFFRES LINUX EMBARQUE

- Les offres de version de Linux embarqué (et Temps Réel) peuvent être rangées dans l'une des 3 catégories suivantes :
 - Les distributions Linux embarqué libres :
 - non TR : μ Clinux, Embedded Debian Project
 - TR dur : Xenomai, RTAI
 - TR mou : PREEMPT-RT
 - autres : eCOS, FreeRTOS...

LINUX EMBARQUE COMMERCIAL



- MontaVista Professional or Carrier Grade or Consumer Electronics Edition :
 - Solution générale (et TR) pour l'embarqué
 - <http://www.mvista.com/>
 - kit d'évaluation disponible (preview kit)
- MontaVista Linux Professional Edition
 - This industry-leading comprehensive embedded operating system and cross development environment is our flagship product. It provides a common source and binary platform across a broad range of processor architectures. The Professional Edition includes a modern OS featuring real-time functionality, multi-process and multi-threaded with extensive bundled software components including rich networking.

LINUX EMBARQUE COMMERCIAL



- Caractéristiques de MontaVista Professional Edition :
- Board Hardware Support
 - Support for over seventy popular COTS, Evaluation, and Reference boards
 - Support for seven target CPU families with more than 25 CPU variants
- MontaVista Development Environment
 - KDevelop IDE
 - MontaVista Target Configuration Tool
 - MontaVista Library Optimizer Tool
 - Graphical binary and source-level debug
 - Graphical kernel configuration tool
 - Kernel debug (KGDB and hardware debuggers)
 - File system populator

LINUX EMBARQUE COMMERCIAL



- Caractéristiques de MontaVista Professional Edition :
- Rich Complement of target-based Software Components
- Deployable utilities, libraries, drivers, and other run-time components
- Real-time Support
 - MontaVista Linux Preemptible Kernel
 - MontaVista Linux Real-time Scheduler with up to 1024 levels of priority
- Rich Networking
 - Extensive complement of clients and servers
 - Rich support for the TCP/IP Suite
 - Broad support for routing, security, tunneling
 - cPCI backplane networking

LINUX EMBARQUE OPEN SOURCE



- µClinux :
 - Pour processeur 32 bits sans MMU.
 - <http://www.uclinux.org>
- Caractéristiques de µClinux (voir après pour plus de détails) :
 - Lineo's uClinux is the ideal OS for non-MMU microprocessors and high-volume embedded systems featuring posix-4, real-time functions, and TCP/IP. uClinux includes a complete TCP/IP stack supporting Ethernet, PPP and SLIP as well as many wireless protocols. uClinux is perfect for remote sensing, monitoring and control applications. And, because uClinux is an open source product, you will never be stuck on a dead end development path.

LINUX EMBARQUE OPEN SOURCE



- Embedded Debian Project :
 - Outil de génération d'un Linux embarqué (OS+FS).
 - <http://www.emdebian.org/>
- Caractéristiques de Embedded Debian Project :
 - The first effort will attempt to capture the current state of the art by exploring the tools and techniques used by other embedded Linux distributions. The primary product of this effort will be the development of a "Guide to Embedding Debian", a comprehensive guide to getting the most (or least, depending on how you look at it) out of Debian for embedded systems.

LINUX EMBARQUE OPEN SOURCE



- **Caractéristiques de Embedded Debian Project :**
- The second effort involves determining the best ways to extend Debian's reach into the embedded space. This will involve things such as:
 - Creating a set of build-tools for embedded system developers to easily package just the parts they need. Emdebsys and ipkg seem to be the basis to work from.
 - Specifying new packages tailored specifically for embedded systems
 - Ensuring new embedded packages work across as many platforms as possible
 - Involvement in Linux standardization activities involving embedded Linux.
 - Work with debian proper to integrate embedded requirements into the debian infrastructure.
- EmDebSys a system for the configuration and generation of both a Linux kernel *AND* an operating system (i.e. root filesystem). EmDebSys is being designed to assist embedded Linux developers in configuring and generating small (1 to 10Mb) Linux target systems (ARM, PowerPC, SPARC, Intel x86, Alpha and Motorola 680x0).

LE CHOIX D 'UN LINUX EMBARQUE

- Le choix est à faire en fonction de ses compétences en interne et des TTM à respecter.
- Choisir un linux embarqué commercial est rassurant. Cela a aussi un coût.

PARTIE 3 : BILAN

LE CHOIX D'UN LINUX EMBARQUE

Complexité de mise en œuvre maximale

LFS (Linux From Scratch)

μ Clinux

Embedded Debian Project

Montavista

Metrowerks/Creation Suite for Linux

LynuxWorks/Bluecat

Complexité de mise en œuvre minimale



PARTIE 4 :

BOITE A OUTILS LIBRES POUR LINUX EMBARQUE

BOITE A OUTILS LINUX EMBARQUE

- Il existe un ensemble de logiciels libres utilitaires (projets) qui facilitent la conception logicielle d'un système sous Linux embarqué.
- On utilise dans ce cadre des utilitaires à faible empreinte mémoire adaptés à l'embarqué. Ils possèdent leur équivalent sous Linux standard.
- Il convient de connaître la boîte à outils libres pour concevoir son système Linux embarqué...

COMPILATEURS CROISES

Outil	Adresse Internet	Description
gcc	http://www.gnu.org/software/gcc/	Compilateur C, C++, Objective-C, Fortran...
crosstool	http://www.kegel.com/crosstool/	Générateur automatique d'un compilateur C, C++ et la bibliothèque <i>glibc</i> pour processeurs alpha, ARM, x86, IA64, MIPS, PowerPC, SH4...
Buildroot	http://buildroot.uclibc.org/	Générateur automatique d'un compilateur C, C++ et la bibliothèque <i>μClibc</i> pour processeurs ARM, MIPS, PowerPC ainsi que le <i>root File System</i>
Scratchbox	http://www.scratchbox.org/	Générateur automatique d'un compilateur C, C++ et la bibliothèque <i>glibc</i> ou <i>μClibc</i> pour processeurs ARM, x86, PowerPC, MIPS

COMPILATEURS CROISES

Outil	Adresse Internet	Description
ELDK	ftp://mirror.switch.ch/mirror/eldk/eldk/	Kit <i>Embedded Linux Development Kit</i> de la société DENX. Contient les compilateurs croisés précompilés pour processeurs PowerPC, ARM, MIPS Documentation : http://www.denx.de/wiki/DULG/Manual
	http://www.codesourcery.com/gnu_toolchains/arm/	Compilateur croisé précompilé pour processeurs ARM
	ftp://ftp.handhelds.org/projects/toolchain/	Compilateur croisé précompilé pour processeurs ARM
	http://www.linux-mips.org/wiki/MIPS_SDE_Installation	Compilateur croisé précompilé pour processeurs MIPS
	http://www.uclinux.org/pub/uClinux/m68k-elf-tools/	Compilateur croisé précompilé pour processeurs m68k, ColdFire et ARM
	http://www.uclinux.org/pub/uClinux/ports/h8/	Compilateur croisé précompilé pour processeurs H8

BIBLIOTHEQUES LIBC

Outil	Adresse Internet	Description
glibc	http://www.gnu.org/software/libc/	<i>libc</i> standard GNU. Peu adaptée à l'embarqué
µClibc	http://www.uclibc.org/	<i>libc</i> pour l'embarqué
newlib	http://sources.redhat.com/newlib/	<i>libc</i> pour l'embarqué
dietlibc	http://www.fefe.de/dietlibc/	<i>libc</i> pour l'embarqué

DEBUG

Outil	Adresse Internet	Description
<code>gdb</code>	http://www.gnu.org/software/gdb/gdb.html	Debugger GNU. Une version « cross compilée » est généralement fournie avec les chaînes de compilation croisée précompilées
DDD	http://www.gnu.org/software/ddd/	<i>Data Display Debugger. Front end à <code>gdb</code></i>

NOYAU LINUX

Outil	Adresse Internet	Description
Noyau Linux	http://www.linux.org/	Noyau Linux standard
Noyau μ Clinux	http://www.uclinux.org/pub/uClinux/uClinux-2.0.x/ http://www.linux-m68k.org/	Noyau μ Clinux pour processeurs m68k
Noyau μ Clinux	http://www.uclinux.org/ports/coldfire/	Noyau μ Clinux pour processeurs ColdFire
Noyau μ Clinux	http://www.uclinux.org/pub/uClinux/ports/arm7tdmi/	Noyau μ Clinux pour processeurs ARM7TDMI
Noyau μ Clinux	http://blackfin.uclinux.org/	Noyau μ Clinux pour processeurs Blackfin
Noyau μ Clinux	http://www.uclinux.org/pub/uClinux/ports/h8/	Noyau μ Clinux pour processeurs H8

NOYAU LINUX

Outil	Adresse Internet	Description
Noyau Linux	http://www.denx.de/en/Software/WebHome	Noyau Linux pour processeurs ARM, PowerPC et MIPS (ELDK)
Noyau Linux	http://www.arm.linux.org.uk/	Noyau Linux pour processeurs ARM
Noyau Linux	http://www.alphalinux.org/	Noyau Linux pour processeurs alpha
Noyau Linux	http://tsukuba.m17n.org/linux-sh/	Noyau Linux pour processeurs superH
Noyau Linux	http://www.linux-mips.org/	Noyau Linux pour processeurs MIPS
Noyau μ Clinux	http://www.niosforum.com	Noyau μ Clinux pour processeurs <i>softcore</i> NIOS II. S'inscrire auparavant
Noyau μ Clinux	http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/	Noyau μ Clinux pour processeurs <i>softcore</i> Microblaze

BOOTLOADERS LINUX

Outil	Adresse Internet	Description
u-boot	http://sourceforge.net/projects/u-boot	<i>Universal Boot. Bootloader Linux pour processeurs PowerPC, ARM, x86, MIPS, Xscale, ColdFire 52x2,PXA... et processeurs softcore MicroBlaze, NIOS I et NIOS II</i>
colilo	http://www.reasonability.net/uclinux/colilo/	<i>Bootloader Linux pour processeurs ColdFire</i>
ppcboot	http://sourceforge.net/projects/ppcboot/	<i>Bootloader Linux pour processeurs PowerPC. Intégré dans u-boot</i>
ARMboot	http://sourceforge.net/projects/armboot	<i>Bootloader Linux pour processeurs ARM</i>
YAMON	http://www.mips.com/sitemap/content/Products/SoftwareTools/content_html/yamon	<i>Bootloader Linux pour processeurs MIPS</i>

BOOTLOADERS LINUX

Outil	Adresse Internet	Description
MRB	http://www.cybertec.com.au/mrb.html	<i>My Right Boot. Bootloader Linux pour processeurs ColdFire</i>
RedBoot	http://ecos.sourceware.org/redboot/	<i>Bootloader Linux pour processeurs ARM, CalmRISC, FR-V, H8, IA32, 68k, Matsushita AM3x, MIPS, NEC V8xx, PowerPC, SPARC, SuperH. S'utilise avec eCos</i>
OpenBIOS	http://www.openbios.info/	<i>BIOS pour carte mère à base de processeurs x86, Alpha et AMD64</i>
FreeBIOS	http://freebios.sourceforge.net/	<i>BIOS pour carte mère à base de processeurs x86</i>
PXELinux	http://syslinux.zytor.com/pxe.php	<i>Bootloader Linux par le réseau Ethernet conforme à la norme PXE pour carte mère de PC à base de processeurs x86</i>
EtherBoot	http://etherboot.sourceforge.net/	<i>Bootloader Linux par le réseau Ethernet pour carte mère de PC à base de processeurs x86</i>

SHELLS ET COMMANDES LINUX

Outil	Adresse Internet	Description
BusyBox	http://busybox.net/	Véritable couteau suisse ! <i>BusyBox</i> est incontournable aujourd'hui
ash	http://www.uclibc.org/http://packages.debian.org/unstable/shells/ash.html	<i>Shell</i> (intégré dans <i>BusyBox</i>)

EDITEURS DE TEXTES

Outil	Adresse Internet	Description
vi	http://busybox.net/	vi est dans <i>BusyBox</i>
ae	http://www.nyangau.fsnet.co.uk/	<i>Andy's Editor</i> . Editeur
e3	http://www.sax.de/~adlibit/	Editeur compatible vi, <i>emacs</i> et <i>Wordstar</i>
elvis	http://elvis.the-little-red-haired-girl.org/	Editeur compatible vi
nano	http://www.nano-editor.org/	Editeur
jove	http://packages.debian.org/unstable/editors/jove	Editeur compatible <i>emacs</i>

OUTILS RESEAU ET CONNECTIVITE IP

Outil	Adresse Internet	Description
boa	http://www.boa.org/	Serveur web. Le serveur web pour l'embarqué. Support des scripts CGI boa+php pour µClinux disponible ici : http://www.menic.org/georges/uClinux/boa-php.html
thttpd	http://www.acme.com/software/thttpd/	<i>tiny/turbo/throttling HTTP server. Serveur web</i>
busybox	http://busybox.net/	<i>BusyBox</i> contient un serveur web
lightTPD	http://lighttpd.net/	Serveur web. Support des scripts CGI
dillo	http://www.dillo.org/	Navigator web. Mode graphique

OUTILS RESEAU ET CONNECTIVITE IP

Outil	Adresse Internet	Description
minimo	http://www.mozilla.org/projects/minimo/	Navigator web. Mode graphique
iproute	http://packages.debian.org/unstable/net/iproute.html	Outil pour la configuration réseau
ntpclient	http://doolittle.faludi.com/ntpclient/	Client NTP
dropbear	http://www.ucc.gu.uwa.edu.au/~Ematt/dropbear/dropbear.html	Client et serveur SSH. Peut être compilé avec µClibc
tinylogin	http://tinylogin.busybox.net/	Outil pour gérer le contrôle d'accès au système embarqué : login, passwd, getty...
udhcp	http://udhcp.busybox.net/	Client et serveur DHCP (intégré dans <i>BusyBox</i>)

INTERFACES GRAPHIQUES

Outil	Adresse Internet	Description
Frame Buffer	http://www.linux.org	Intégré au noyau Linux
DirectFB	http://www.directfb.org/	Environnement graphique tournant au-dessus du <i>Frame Buffer</i>
Nano-X	http://www.microwindows.org/	Environnement graphique tournant au-dessus du <i>Frame Buffer</i> , X Window, SVGAlib. Il existe 2 API (<i>Application Programming Interface</i>) : - Nano-X : API compatible Xlib - Win32 : API compatible Windows Win32
Minigui	http://www.minigui.org/	Environnement graphique tournant au-dessus du <i>Frame Buffer</i>
FLTK	http://fltk.org/	Environnement graphique tournant au-dessus de X11/Xlib avec OpenGL
SDL	http://www.libsdl.org/index.php	Environnement graphique tournant au-dessus du <i>Frame Buffer</i>
OPIE	http://opie.handhelds.org/	Open Palmtop Integrated Environment. Environnement graphique tournant au-dessus du <i>Frame Buffer</i>
GPE	http://gpe.handhelds.org/	<i>GPE Palmtop Environment.</i> Environnement graphique tournant au-dessus du X11

EMULATEURS

Outil	Adresse Internet	Description
qemu	http://qemu.org/	Emulateur pour processeurs x86, ARM, SPARC et PowerPC. Peut émuler un système Linux pour processeurs x86, x86_64 et PowerPC
UML	http://user-mode-linux.sourceforge.net/	Emulateur d'un système Linux pour processeurs x86
Coldfire Emulator	http://www.slicer.ca/coldfire/	Emulateur pour processeurs ColdFire. Peut émuler un système μ Clinux
SkyEye	http://skyeye.sourceforge.net/index.shtml	Emulateur pour processeurs ARM. Peut émuler un système Linux ou μ Clinux
Softgun	http://softgun.sourceforge.net/	Emulateur pour processeurs ARM. Peut émuler un système Linux
SWARM	http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html	Emulateur pour processeurs ARM. Peut émuler un système μ Clinux

GUI

- Mise en œuvre de Java avec la JVM comme OS : Android.
- Il est possible aussi d'utiliser des interfaces graphiques légères :
 - Nano-X.
 - Qt Embedded de Nokia (ex Trolltech) (et dérivés Qtopia, OPIE).
 - Frame buffer.

AUTRES

Outil	Adresse Internet	Description
MTD	http://www.linux-mtd.infradead.org/	<i>Memory Technology Device</i> . Bibliothèque pour gérer la mémoire FLASH. Intégré au noyau Linux
JFFS2	http://sources.redhat.com/jffs2/	<i>Journalling Flash File System, version 2. Système de fichiers pour mémoire FLASH</i>
LTP	http://ltp.sourceforge.net/	<i>Linux Test Project</i> . Suite de tests d'un système Linux

PARTIE 5 :

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Pour bien concevoir son système embarqué libre, il convient de respecter quelques points importants.
- Il s'agit de règles de bon sens !

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Le système d'exploitation est Linux. Il faut donc choisir un processeur 32 bits supporté par Linux ou μ Clinux (avec ou sans MMU).
- Il faut choisir des composants électroniques supportés par Linux et non l'inverse. On n'aura à développer que les drivers spécifiques de son application.

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Très tôt dans le processus de conception, on utilisera une carte d'évaluation du commerce utilisant le même processeur que celui de son design.
- Il est même possible d'utiliser cette carte d'évaluation comme carte finale de son design en rajoutant éventuellement des cartes filles pour sa partie spécifique.
- Cela est d'autant plus facile que l'on utilise des cartes au format industriel : PC/104, EBX, ETX...

BIEN CONCEVOIR SON SYSTÈME EMBARQUÉ LIBRE

- Il serait étonnant de ne pas avoir le portage de Linux embarqué pour la carte d'évaluation choisie !
- Il est alors possible de développer la partie logicielle sur la carte d'évaluation en ajoutant les cartes filles spécifiques. Cela est généralement simplifié par la présence de connecteurs d'E/S permettant de s'interfacer sur le bus du processeur.
- Il est fourni généralement les schémas électroniques de la carte d'évaluation, ce qui permet une reprise de CAO pour la carte finale.

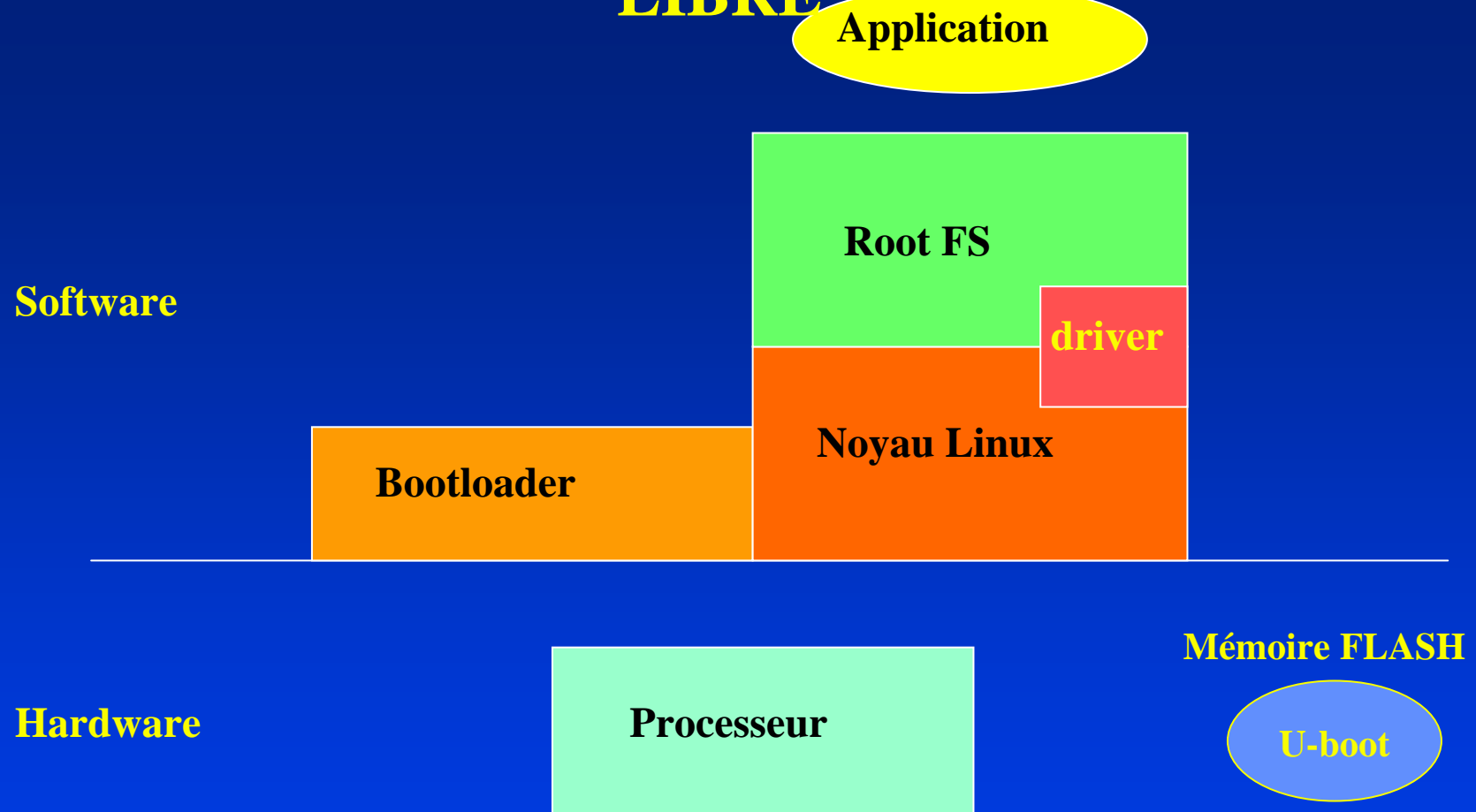
BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Il ne faut pas hésiter à s'abonner et poser des questions dans les forums adéquats. La réponse est rapide.
- Il ne faut pas hésiter à suivre des formations pour gagner du temps. C'est un investissement pour l'avenir !
- L'expérience acquise dans le libre pour l'embarqué peut déboucher vers d'autres marchés...

BIEN CONCEVOIR SON SYSTÈME EMBARQUÉ LIBRE

- Il faut privilégier la solution Linux embarqué la plus simple.
- Au minimum, pour concevoir son système Linux embarqué, on a besoin :
 - D'un bootloader Linux : u-boot.
 - Du noyau Linux.
 - D'un système de fichiers root : busybox.

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE



BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- U-boot : on adaptera le bootloader à sa carte cible.
- Noyau Linux : le noyau Linux sera configuré et éventuellement adapté à sa carte cible puis cross-compilé.
- Bibliothèque libc : on utilisera une bibliothèque adaptée : μ Clibc...
- Busybox : le « couteau suisse » sera configuré puis cross-compilé.

CHAPITRE 5 :

LE TEMPS REEL SOUS LINUX

INTRODUCTION AU TEMPS REEL

PARTIE 1 : INTRODUCTION

TEMPS REEL MOU

- Un système d'exploitation est dit Temps Réel (dur) s'il est capable de répondre à des sollicitations ou événements (internes ou externes) dans un temps maximum.
- On parle de Temps Réel mou (*Soft Real Time*) quand les événements traités trop tardivement ou perdus sont sans conséquence catastrophique pour la bonne marche du système. On ne garantit qu'un pourcentage **moyen** d'utilisation du temps CPU.
- Les systèmes à contraintes *souples* ou *molles* (*soft real time*) acceptent des variations dans le traitement des données de l'ordre de la demi-seconde (ou 500 ms) ou la seconde.

TEMPS REEL MOU

- On peut citer l'exemple des systèmes multimédia : si quelques images ne sont pas affichées, cela ne met pas en péril le fonctionnement correct de l'ensemble du système.
- Ces systèmes se rapprochent fortement des systèmes d'exploitation classiques à temps partagé.
- Ils garantissent un temps moyen d'exécution pour chaque tâche (un débit, une Bande Passante).
- On a ici une répartition **égalitaire** du temps CPU entre processus.

TEMPS REEL DUR

- On parle de Temps Réel dur (*Hard Real Time*) quand les événements traités trop tardivement ou perdus provoquent des conséquences catastrophiques pour la bonne marche du système (perte d'informations cruciales, plantage...).
- Les systèmes à contraintes *dures* (*hard real time*) ne tolèrent qu'une gestion stricte du temps est nécessaire afin de conserver l'intégrité du service rendu.
- On citera comme exemples les contrôles de processus industriels sensibles comme la régulation des centrales nucléaires ou les systèmes embarqués utilisés dans l'aéronautique.

TEMPS REEL DUR

- Ces systèmes garantissent un temps maximum d'exécution pour chaque tâche.
- On a ici une répartition **totalitaire** du temps CPU entre tâches.
- On peut dire qu'un système temps réel doit être prévisible (*predictible* en anglais), les contraintes temporelles pouvant s'échelonner entre quelques micro-secondes (μ s) et quelques secondes.

TEMPS REEL DUR

- Les systèmes à contraintes dures doivent répondre à trois critères fondamentaux :
 - Le déterminisme *logique* : les mêmes entrées appliquées au système doivent produire les mêmes effets.
 - Le déterminisme *temporel* : un tâche donnée doit obligatoirement être exécutée dans les délais impartis, on parle d'*échéance*.
 - La *fiabilité* : le système doit être disponible. Cette contrainte est très forte dans le cas d'un système embarqué car les interventions d'un opérateur sont très difficiles voire même impossibles. Cette contrainte est indépendante de la notion de temps réel mais la fiabilité du système sera d'autant plus mise à l'épreuve dans le cas de contraintes dures.

LINUX ET LE TEMPS REEL

- Linux n'est pas un système d'exploitation Temps Réel (dur) car :
 - Le noyau Linux possède de longues sections de code où tous les événements extérieurs sont masqués (non interruptible).

ET

- Le noyau Linux n'est pas préemptible durant toute l'exécution d'un appel système (structure monolithique) par un processus et ne le redevient qu'en retour d'appel système (mode user).
- Le noyau Linux n'est pas préemptible durant le service d'une interruption (ISR). La routine ISR acquitte l'interruption puis programme un « Bottom Half » (BH) pour le traitement des données. Le BH est exécuté de façon non préemptif immédiatement avant de retourner en mode user.

LINUX ET LE TEMPS REEL

- Linux n'est pas un système d'exploitation Temps Réel (dur) car :
 - En cas d'occurrence d'une interruption durant l'exécution d'un appel système en mode noyau, le BH programmé par l'ISR (et éventuellement les autres BH des autres ISR) ne sera exécuté qu'à la fin de l'exécution complète de l'appel système d'où un temps de latence important et non borné fatal à un système Temps Réel !

LINUX ET LE TEMPS REEL

- Linux n'est pas un système d'exploitation Temps Réel (dur) car :
 - L'ordonnanceur de Linux essaye d'attribuer de façon équitable le CPU à l'ensemble des processus (ordonnancement de type *old aging* mise en œuvre pour favoriser l'accès CPU aux processus récents). C'est une approche égalitaire. Un ordonnanceur Temps Réel donnera toujours la main à la tâche de plus forte priorité prête. C'est ici une approche plus totalitaire.

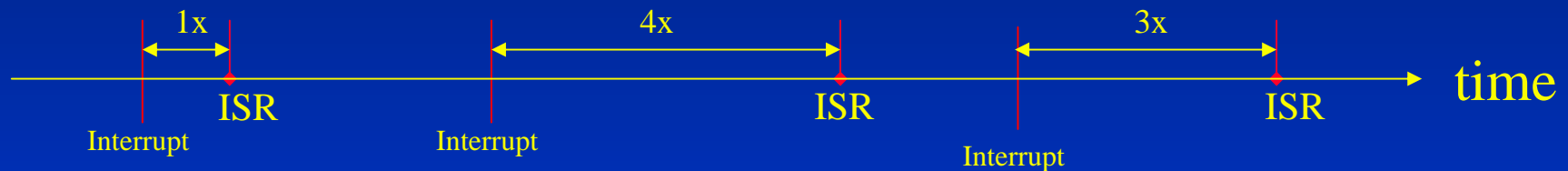
LINUX ET LE TEMPS REEL

- Le noyau Linux standard peut être considéré comme Temps Réel (mou) par définition si l'on travaille avec une réactivité de l'ordre de la centaine de ms ou plus.
- Il existe des solutions Linux Temps Réel dur pour une réactivité de quelques dizaines de μ s...
- Il existe des solutions Linux Temps Réel mou par application de patches dits préemptifs sur un noyau Linux standard pour une réactivité de quelques centaines de μ s...

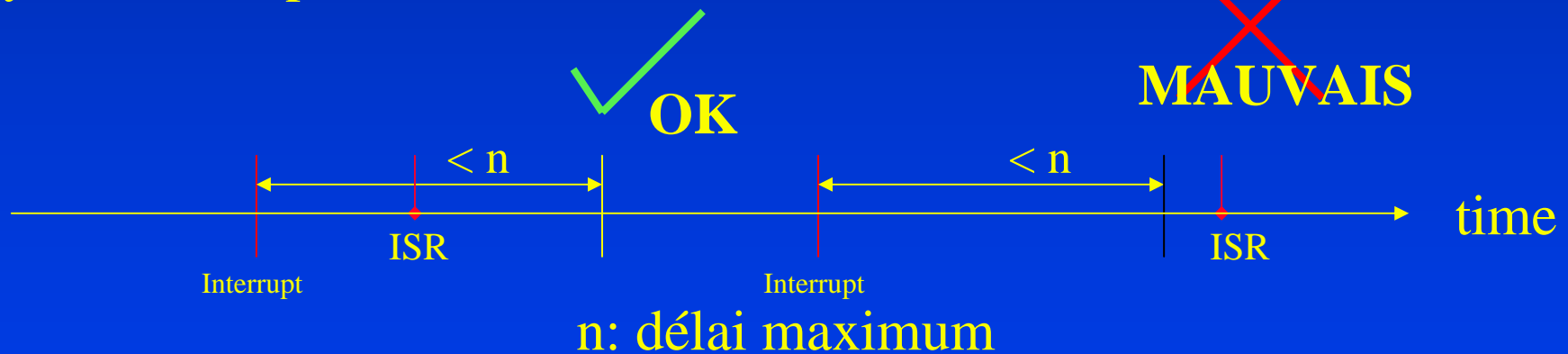
LINUX ET LE TEMPS REEL

- Traitement des interruptions et ISR (*Interrupt Sub Routine*) :

Linux



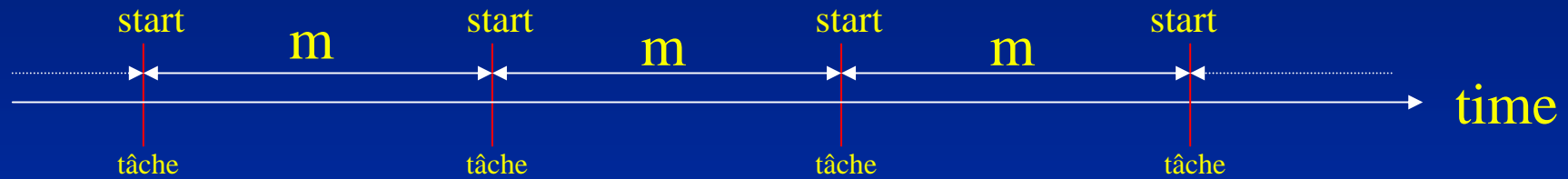
Système Temps Réel (RTOS)



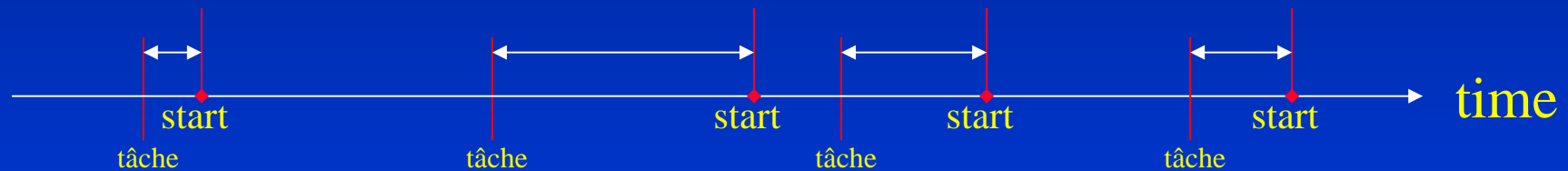
LINUX ET LE TEMPS REEL

- Traitement des tâches périodiques :

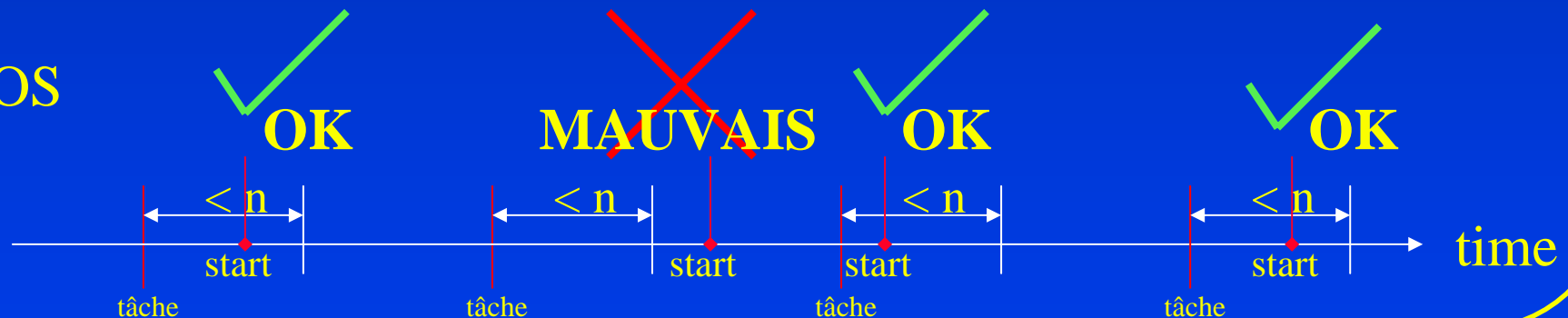
Idéalement



Linux



RTOS



UNE PETITE EXPERIENCE

- Génération d'un signal périodique sur une broche du port parallèle.
- Le signal généré sur la broche 2 (bit D0) du port parallèle est théoriquement un signal périodique carré de demi-période $T/2$ de 50 ms.
- On observe à l'oscilloscope le signal suivant sur un système non chargé (AMD Athlon 1500+).

UNE PETITE EXPERIENCE

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <asm/io.h>

#define LPT 0x378

int ioperm();

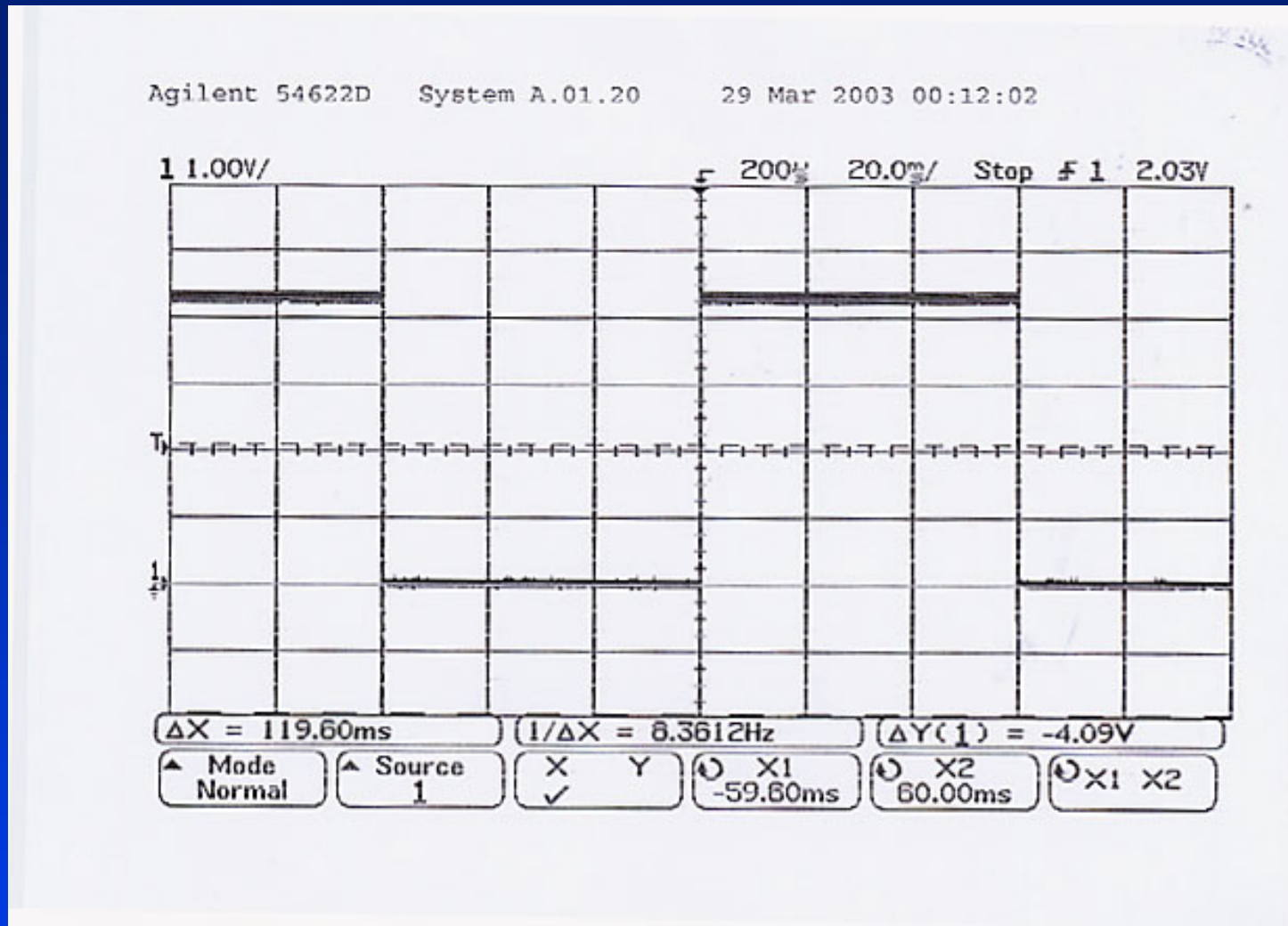
int main(int argc, char **argv)
{
    setuid(0);
    if (ioperm(LPT, 1, 1) < 0) {
        perror("ioperm()");
        exit(-1);
    }
}
```

UNE PETITE EXPERIENCE

```
while(1) {  
    outb(0x01, LPT);  
    usleep(50000);  
  
    outb(0x00, LPT);  
    usleep(50000);  
}  
return(0);  
}
```

Programme square (fichier C square.c)

UNE PETITE EXPERIENCE

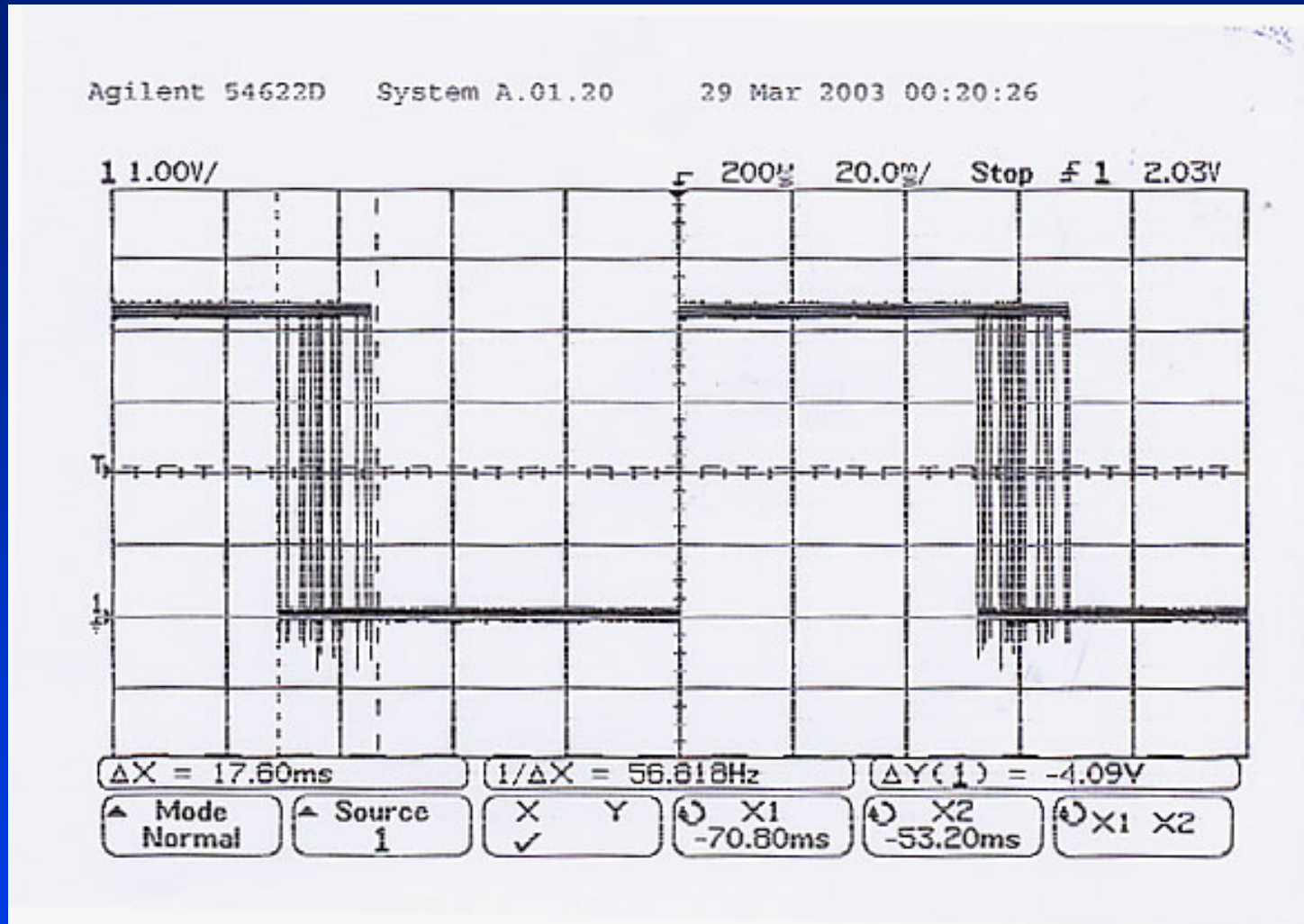


Génération d'un signal carré sous Linux non chargé

UNE PETITE EXPERIENCE

- On remarque que l'on n'a pas une période de 100 ms mais de 119,6 ms dû au temps supplémentaire d'exécution des appels système.
- Dès que l'on stresse le système (écriture répétitive sur disque d'un fichier de 50 Mo), on observe le signal suivant :

UNE PETITE EXPERIENCE



Génération d'un signal carré sous Linux chargé

UNE PETITE EXPERIENCE

- On observe maintenant une gigue (*jitter*) sur le signal généré. La gigue maximale sur la durée de l'expérience est de 17,6 ms.
- La forme du signal varie maintenant au cours du temps, n'est pas de forme carrée mais rectangulaire. LINUX n'est donc plus capable de générer correctement ce signal.
- Il faut noter aussi que le front montant sur la figure précédente apparaît sans gigue car il a servi comme front de synchronisation de l'oscilloscope. La gigue observée est donc à voir comme la contribution de la gigue sur front montant et sur front descendant.
- Si l'on diminue la valeur de la demi-période, la gigue devient aussi importante que cette dernière et dans ce cas, Linux ne génère plus aucun signal !

LINUX ET LES NORMES POSIX

- La complexité des systèmes et l'interopérabilité omniprésente nécessitent une standardisation de plus en plus grande tant au niveau des protocoles utilisés que du code source des applications. Même si elle n'est pas obligatoire, l'utilisation de systèmes conformes à *POSIX* est de plus en plus fréquente.
- *POSIX* est l'acronyme de *Portable Operating System Interface* ou interface portable pour les systèmes d'exploitation.
- Cette norme a été développée par l'IEEE (*Institute of Electrical and Electronic Engineering*) et standardisée par l'ANSI (*American National Standards Institute*) et l'ISO (*International Standards Organisation*).

LINUX ET LES NORMES POSIX

- Le but de POSIX est d'obtenir la portabilité des logiciels au niveau de leur code source.
- Un programme qui est destiné à un système d'exploitation qui respecte POSIX doit pouvoir être adapté à moindre frais sous n'importe quel autre système POSIX. En théorie, le portage d'une application d'un système POSIX vers un autre doit se résumer à une compilation des sources du programme.
- POSIX a initialement été mis en place pour les systèmes de type UNIX mais d'autres systèmes d'exploitation comme *Windows NT* sont aujourd'hui conformes à POSIX.

LINUX ET LES NORMES POSIX

- Le standard POSIX est divisé en plusieurs sous-standards dont les principaux sont les suivants :
 - IEEE 1003.1-1990 : POSIX Partie 1 : Interface de programmation (API) système. Définition d'interfaces de programmation standards pour les systèmes de type UNIX, connu également sous l'appellation ISO 9945-1. Ce standard contient la définition de ces fonctions en langage C.
 - IEEE 1003.2-1992 : Interface applicative pour le *shell* et applications annexes. Définit les fonctionnalités du shell et commandes annexes pour les systèmes de type UNIX.
 - IEEE 1003.1b-1993 : Interface de programmation (API) temps réel. Ajout du support de programmation temps réel au standard précédent. On parle également de POSIX.4.
 - IEEE 1003.1c-1995 : Interface de programmation (API) pour le multithreading.

LINUX ET LA NORME POSIX 1003.1b

- La norme POSIX 1003.1b définit la notion de Temps Réel pour un système d'exploitation à des fins de normalisation :
“Realtime in operating systems: the ability of the operating system to provide a required level of service in a bounded response time ”
- Cela implique pour le système d'exploitation :
 - D'avoir un ordonnancement des tâches préemptif (avec priorité).
 - De coller en mémoire les pages virtuelles.
 - De supporter des signaux Temps Réel.
 - D'avoir des mécanismes de communication inter-processus IPC (*Inter Processus Communication*) performants.
 - D'avoir des timers Temps Réel.

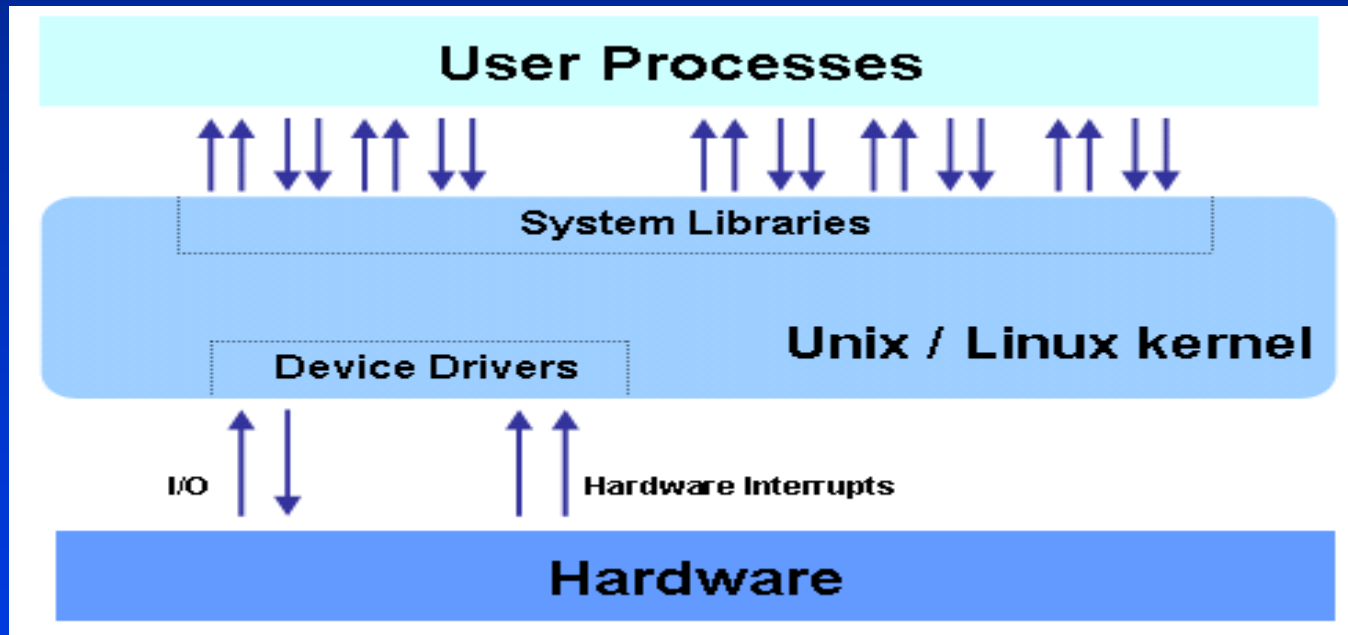
LINUX ET LA NORME POSIX 1003.1b

- Linux ne supporte que partiellement la norme POSIX 1003.1b et ne peut être considéré en l'état comme un système Temps Réel.
 - Appels système : *mlock()*, *setsched()*.
- Il convient donc de modifier Linux pour le rendre Temps Réel et conforme à la norme POSIX 1003.1b.

EXTENSION TEMPS REEL POUR LINUX

- Implémentation du noyau Linux standard :
 - Pas de support du Temps Réel.
 - Séparation entre le matériel et les processus Linux.

– ...



Linux standard

EXTENSION TEMPS REEL POUR LINUX

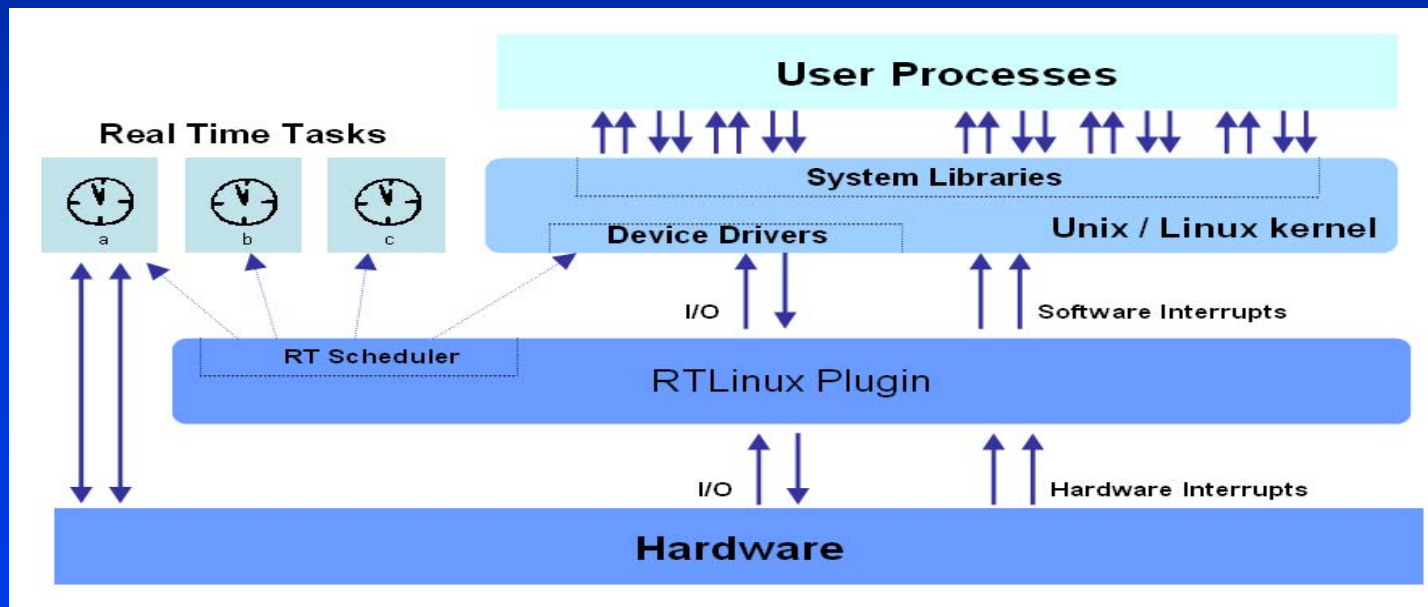
- Solution 1 pour une extension Temps Réel mou de Linux :
 - Modification du noyau Linux par application de patches pour améliorer les contraintes Temps Réel : dévalider les interruptions le moins longtemps possible, appeler l'ordonnanceur le plus souvent possible (fonction *schedule()* du noyau), en retour d'interruption par exemple, réduction des temps de latence. Ces modifications ne transforment pas Linux en noyau temps réel dur mais permettent d'obtenir des résultats satisfaisants dans le cas de contraintes temps réel molles.
 - Cette technologie est disponible avec le projet open source PREEMPT-RT et elle est également supportée commercialement par divers éditeurs spécialisés comme MontaVista, TimeSys.

EXTENSION TEMPS REEL POUR LINUX

- Solution 2 pour une extension Temps Réel dur de Linux :
 - Ajout d'un deuxième ordonnanceur TR de tâches et considérer le noyau Linux et ses processus comme tâche de fond. Plus difficile que la première solution.
 - Cette technique permet de mettre en place des systèmes temps réel durs.
 - Utilisé dans les projets Xenomai et RTAI.

EXTENSION TEMPS REEL POUR LINUX

- Solution 2 pour une extension Temps Réel dur de Linux :
 - Ajout d'une couche d'abstraction entre le matériel et le noyau Linux.
 - Définition de tâches Temps Réel.
 - Pas de séparation entre le matériel et les tâches Temps Réel.



EXTENSION TEMPS REEL POUR LINUX

EVOLUTION DE LA PRÉEMPTIVITÉ DU NOYAU LINUX

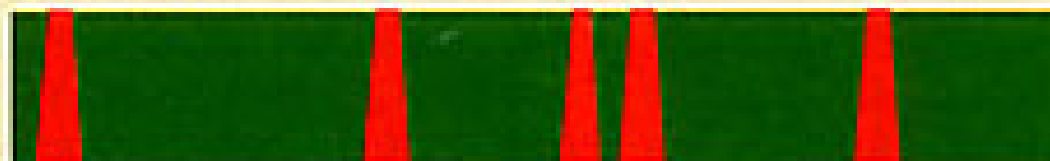
Pas de
préemption
(Linux 2.4)



Préemption
volontaire



Noyau
préemptible
(Linux 2.6)



Préemption
temps réel



Sections préemptibles

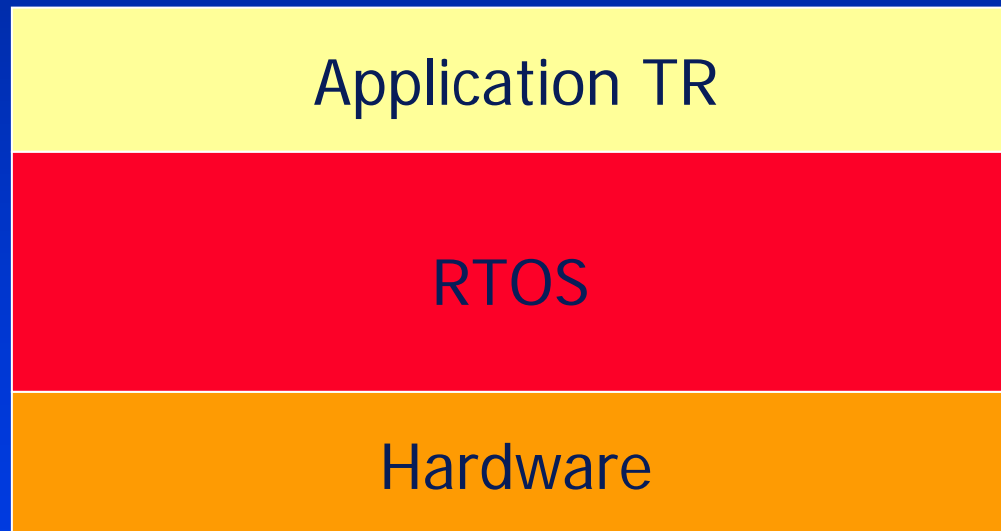


Sections non préemptibles

L'amélioration des temps de réponse de Linux passe notamment par une réduction de la longueur des sections non préemptibles du noyau.

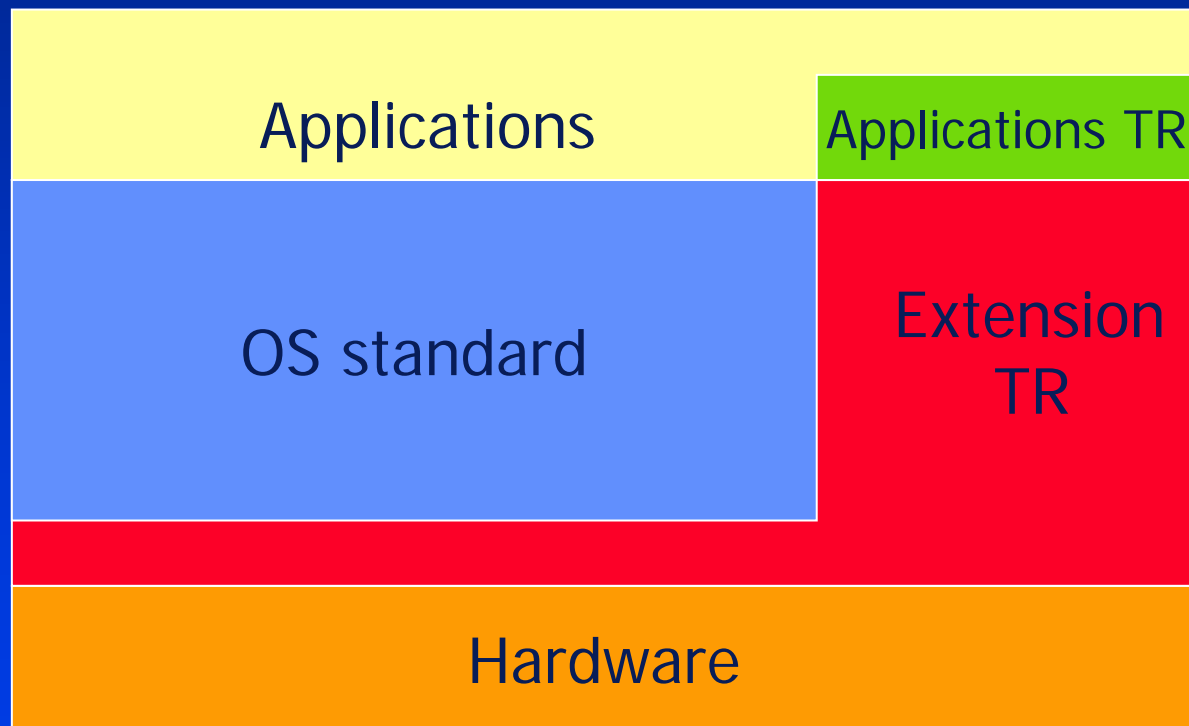
VERITABLE RTOS

- Avantages : simplicité, monolithique, fait pour le TR, petit overhead.
- Inconvénients : fonctionnalités limitées.
- Exemples : VxWorks, QNX, pSOS, VRTX, μ C/OS II, eCOS, FreeRTOS...



RTOS PAR EXTENSION

- Avantages : de nombreuses fonctionnalités, coopération entre tâches TR et processus non TR.
- Inconvénients : n'est pas un vrai RTOS monolithique.
- Exemples : Xenomai, RTAI...



PARTIE 2 : LES PATCHS PREEMPTIFS

PATCH DU NOYAU

- Il existe deux principaux patches permettant d'améliorer les performances du noyau Linux 2.4 au niveau du temps de latence en diminuant la taille des sections du noyau non préemptibles :
 - Patch Preempt Kernel
 - Patch Low Latency

PATCH DU NOYAU

- Le patch Preempt Kernel est maintenu par Robert M. Love et soutenu par MontaVista :

<http://www.tech9.net/rml/linux>

- Le principe du patch est de rendre le noyau totalement préemptible et de protéger les données du noyau par des mutexs ou spinlocks.
- A chaque fois qu'un événement apparaît et rend un processus de plus forte priorité prêt, le noyau préempte le processus courant et exécute le processus de plus forte priorité.

PATCH DU NOYAU

- Ce mécanisme de préemption ne doit pas être mis en oeuvre :
 - lors du service d'une interruption (interruption handling)
 - lors de la prise d'un verrou spinlock, writelock, ou readlock (verrou utilisé pour le SMP : Symmetric Multi Processing)
 - lors de l'exécution de l'ordonnanceur lui-même
 - lors de l'exécution du processing Bottom Half (BH)
- Voir <http://www.linuxdevices.com/articles/AT4185744181.html> pour plus de détails.

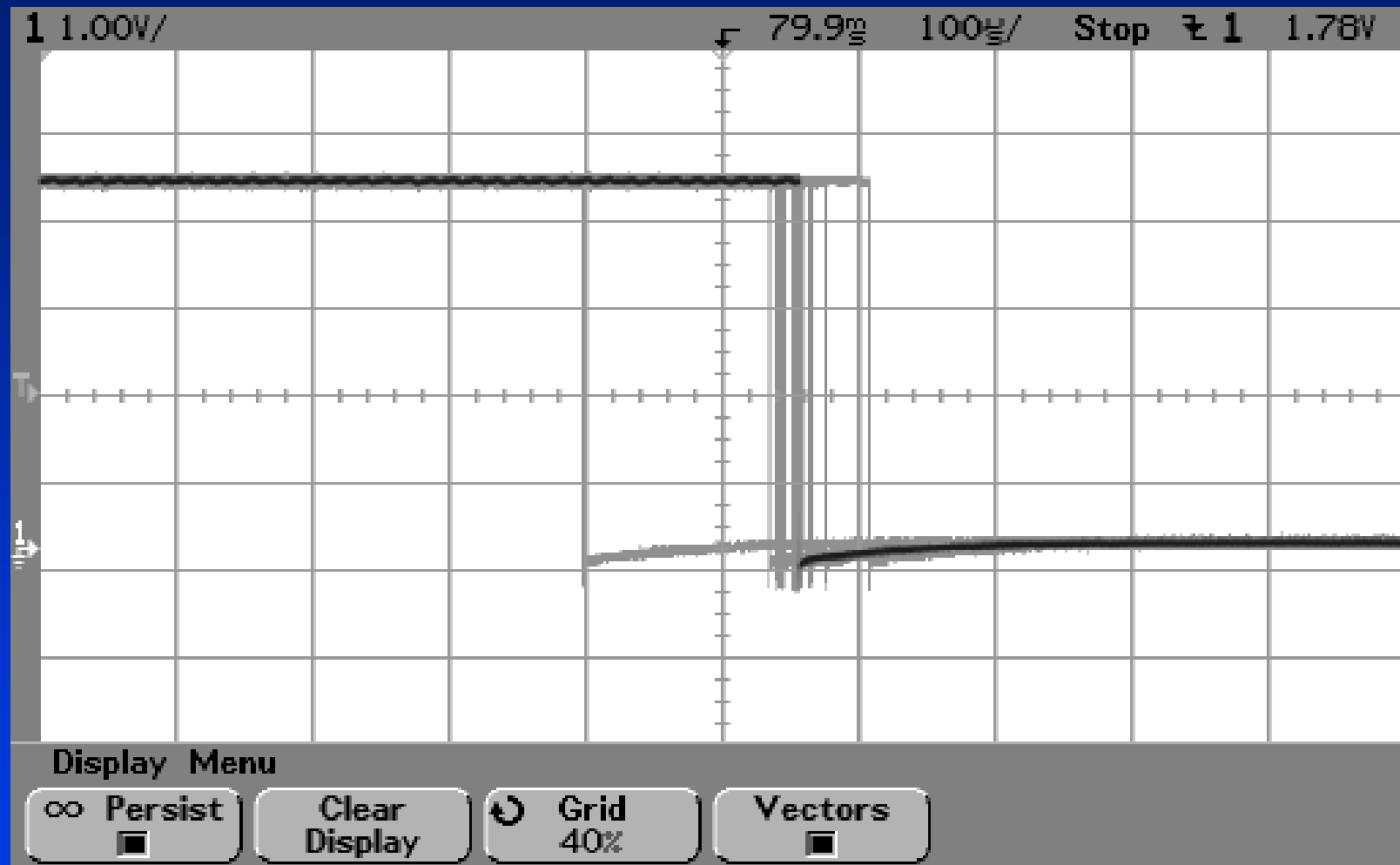
PATCH DU NOYAU

- Le patch Low Latency est maintenu par Andrew Morton : <http://www.zip.com.au/~akpm/linux/schedlat.html>
- Le principe est un peu différent car au lieu d'opter pour une stratégie systématique du noyau tout préemptif, les développeurs du patch ont préféré effectuer une analyse du noyau afin d'ajouter des points de préemption obligatoire (appel de `schedule()`) subtilement placés dans les sources du noyau afin de casser des boucles d'attente trop longues.

PETITE EXPERIENCE. SUITE

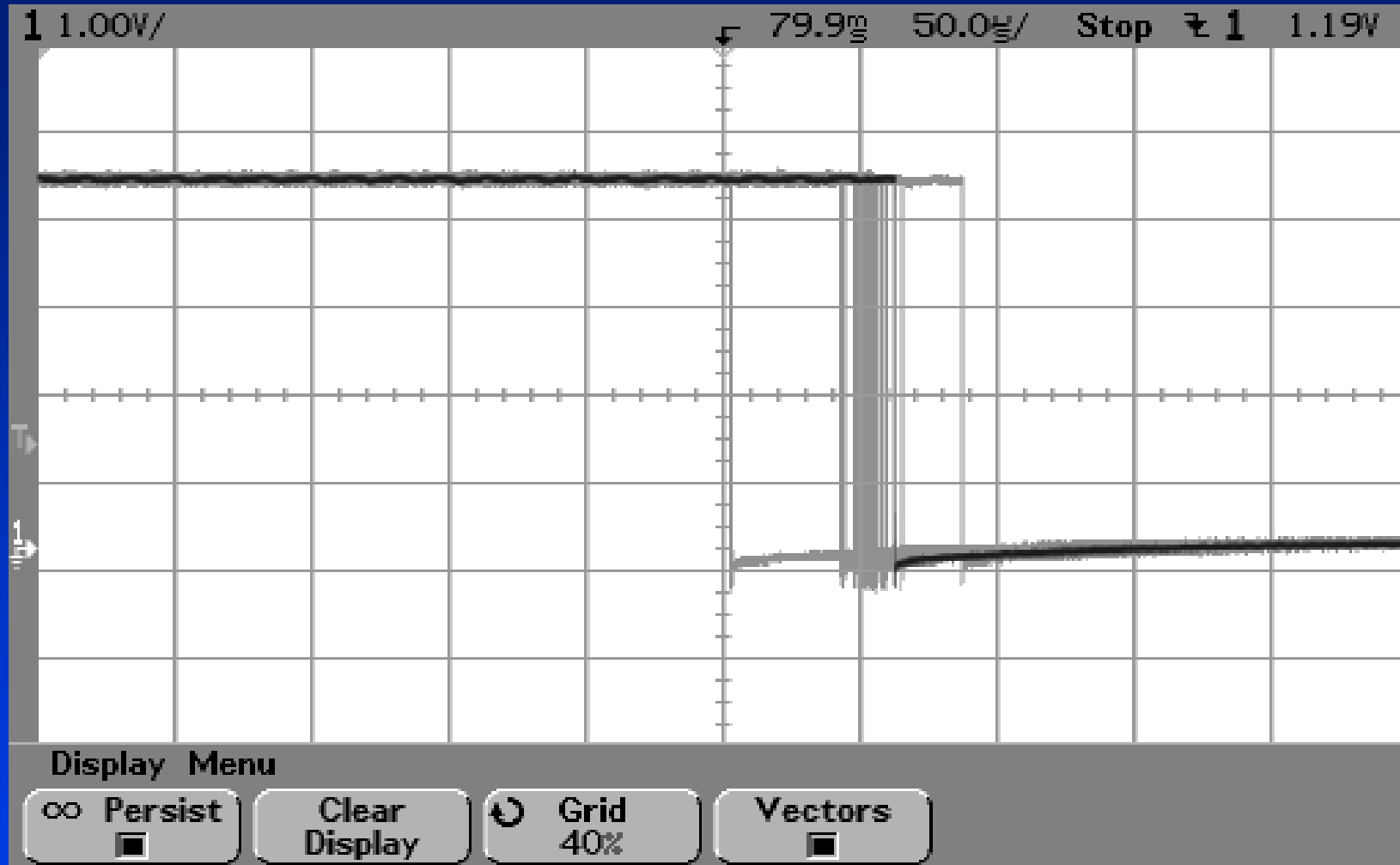
- Mesure effectuée à l'oscilloscope lors de l'utilisation du programme square décrit précédemment.
- Dans le cas du noyau 2.4.20 modifié par le patch *Preempt Kernel* et subissant la même charge que pour les autres mesures, nous obtenons la courbe suivante, indiquant une latence maximale légèrement supérieure à 200 μ s.
- Dans le cas du patch *Low Latency*, nous obtenons un meilleur résultat avec une latence maximale d'environ 80 μ s.

PETITE EXPERIENCE. SUITE



Exécution de square avec le patch Preempt Kernel

PETITE EXPERIENCE. SUITE



Exécution de square avec le patch Low Latency

CONCLUSION

- Les patches précédents permettent d'améliorer les temps de latence sur le noyau Linux standard mais le concept se rapproche plus d'une amélioration de la qualité de service que du temps réel dur.
- La réactivité est maintenant de l'ordre de quelques dizaines à quelques centaines de μs au lieu de quelques dizaines à quelques centaines de ms voire plus pour un noyau Linux standard !
- Des mesures faites par Metrowerks à l'aide du Latency Benchmark de Systems Software Labs montrent que dans 99,5 % des cas le temps de latence est inférieur à 200 μs pour les 2 patches (voir le whitepaper *Linux as a Real-Time operating System* de Metrowerks).

PARTIE 3 :

LES OFFRES LINUX TEMPS REEL

LES OFFRES LINUX TEMPS REEL

- Les offres de version de Linux embarqué et Temps Réel peuvent être rangées dans l'une des 2 catégories suivantes :
 - Les distributions Linux Temps Réel commerciales.
 - Les distributions Linux Temps Réel libres.

LINUX TEMPS REEL COMMERCIAL

- Montavista Professional
- Lineo-Metrowerks-Motorola/Creation Suite for Linux
- LynuxWorks/BlueCat RT
- TimeSys/Linux RTOS Professional or Standard Edition

LINUX TEMPS REEL OPEN SOURCE

- Xenomai
http://xenomai.org/index.php/Main_Page
- RTAI : Real Time Application Interface
<http://www.aero.polimi.it/~rtai/>
- eCOS
<http://sources.redhat.com/ecos/>
- FreeRTOS
<http://www.freertos.org/>

TEMPS REEL COMMERCIAL

- Il y a toujours les solutions TR commerciales non Linux :
 - VxWorks
 - pSOS
 - QNX
 - LynxOS
 - μ C/OS II
 - ...

TEMPS REEL COMMERCIAL : VxWorks ET pSOS

- Solution commerciale TR non Linux. Noyau TR. WindRiver Systems.
 - <http://www.wrs.com>
- Un des leaders dans le domaine du TR et de l'embarqué...
- **VxWorks :**
 - Scalable (simple to complex designs)
 - Reliable (mission-critical applications, ABS)
 - CPU's PowerPc, 68K, CPU32, ColdFire, MCORE, 80x86 and Pentium, i960, ARM and StrongARM, MIPS, SH, SPARC, NECV8xx, M32 R/D, RAD6000, ST 20, TriCore)
 - Graphic Development Platform
 - Cross-development
 - Support/Documentation
 - POSIX 1003.1b compliant
 - **Networking**
 - Tornado II embedded development platform

TEMPS REEL COMMERCIAL : QNX

- Solution commerciale TR non Linux. Système TR. QNX software Systems.
 - <http://www.qnx.com>
- QNX :
 - Highly reliable
 - all generic x86 based processors(386+)
 - Scalable (modules)
 - Deterministic
 - "QNX Neutrino™ real-time OS, "the most advanced RTOS on the market"
 - Networking
 - Graphical development tools and debugger
 - Visual design tools (C code form cut and paste)

TEMPS REEL COMMERCIAL : LynxOS



- Solution commerciale TR compatible Linux. Système TR. LynuxWorks Systems (ex Lynx).
 - <http://www.lynxos.com/>
- LynxOS is unique in the real-time embedded software marketplace. It is a hard RTOS that combines performance, reliability, openness, and scalability together with patented technology for real-time event handling. Flexible scalability makes the LynxOS well suited for applications ranging from large and complex switching systems down to small highly embedded products. **LynxOS is binary compatible with the BlueCat Linux**, enabling users to take advantage of the best configuration for their needs. In addition, LynuxWorks also supports traditional UNIX and Java and supports processors from Intel, Motorola, and MIPS. LynxOS offers users a choice of software application interfaces, a large number of development tools, scalability and memory efficiency which reflect the many years of expertise LynuxWorks has in the real-time embedded systems market.

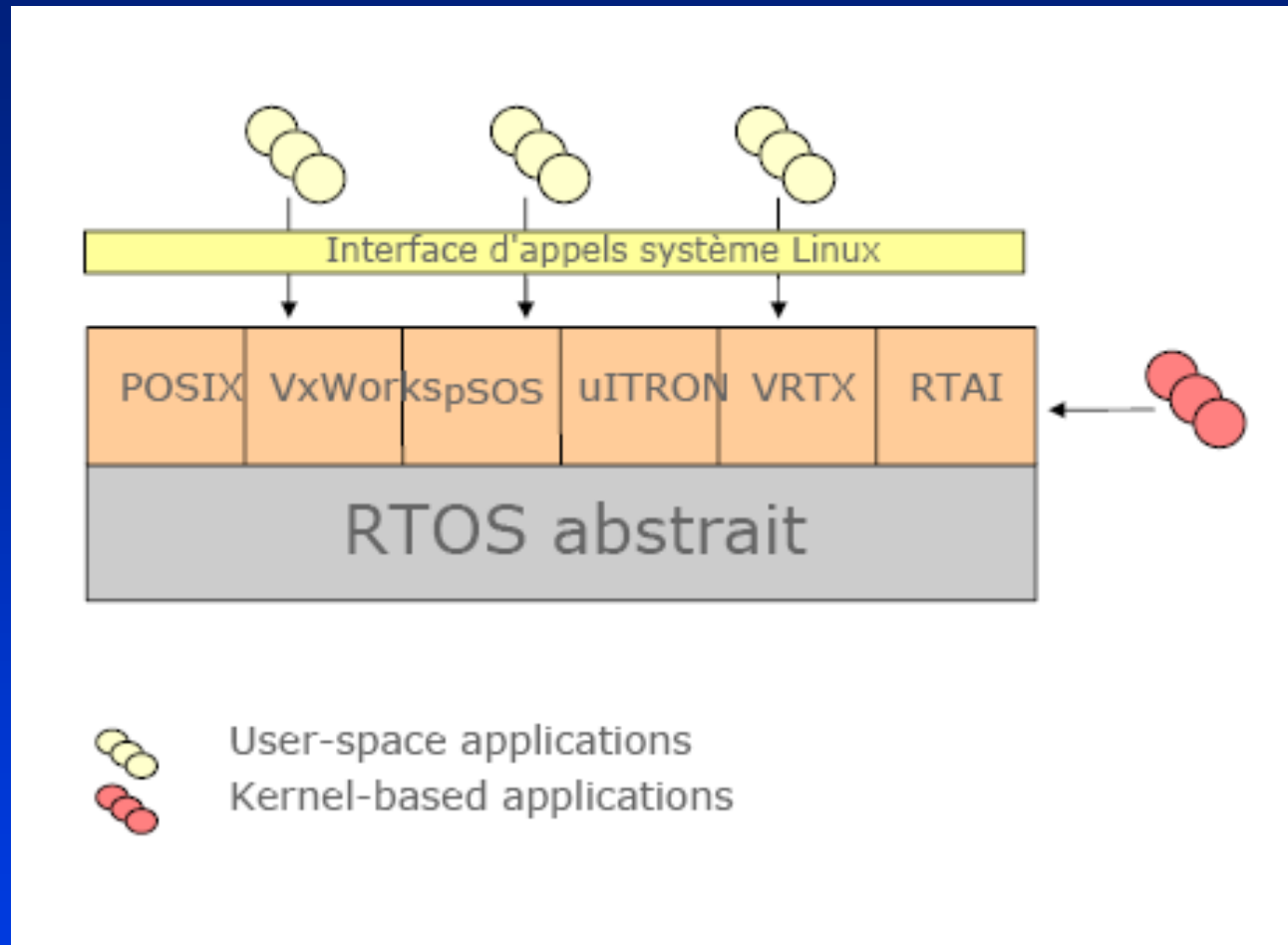
LINUX TEMPS REEL OPEN SOURCE : RTAI

- Solution libre d'extension TR de Linux. Système TR. Université de Milan en Italie.
- <http://www.aero.polimi.it/~rtai/>
- Mise en place d'une couche d'abstraction
- Mise en service sous forme de modules Linux :
 - 3 entités/modules de base :
 - Dispatcher d'interruptions entre tâches TR ou noyau Linux.
 - Ordonnanceur TR.
 - FIFOS de communication.

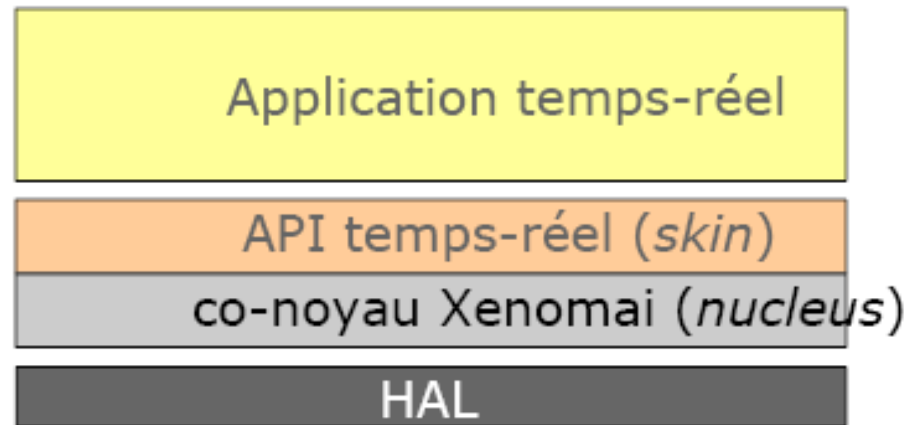
LINUX TEMPS REEL OPEN SOURCE : XENOMAI



- Mise en place coeur de noyau Temps Réel générique
- Emulation d'APIs propriétaires (VxWorks, pSOS, VRTX, uITRON)
- Architectures supportées :
 - ARM, Blackfin, x86, IA64, PowerPC32, PowerPC64, NIOS II
- Programmation en mode :
 - Espace kernel
 - Espace utilisateur

LINUX TEMPS REEL OPEN SOURCE : XENOMAI



LINUX TEMPS REEL OPEN SOURCE : XENOMAI



-  couche API (POSIX, pSOS+, VRTX, VxWorks, uITRON ...)
-  RTOS abstrait

LINUX TEMPS REEL OPEN SOURCE : XENOMAI

- Xenomai est sûrement la solution Temps Réel dur sous Linux la plus prometteuse.
- Elle est à privilégier...

PARTIE 4 : PRESENTATION DE RTLinux OUTDATED

HISTOIRE DE RTLinux

- RTLinux a été développé originellement par un chercheur de l'université de New Mexico avec l'aide d'un étudiant : Victor Yodaiken et Michael Barabanov.
- D'abord sous licence GPL, un brevet (US Patent No. 5,995,745) a été déposé sur le principe de fonctionnement de RTLinux, ce qui est incompatible avec la notion de logiciel libre. Une entreprise privée FSMLabs a été créée pour distribuer RTLinux.
- Devant le tollé général, FSMLabs décide de distribuer une version GPL OpenRTLinux et une version commerciale RTLinux/PRO plus complète.
- RTLinux a été racheté par WindRiver. Ne semble plus supporté...

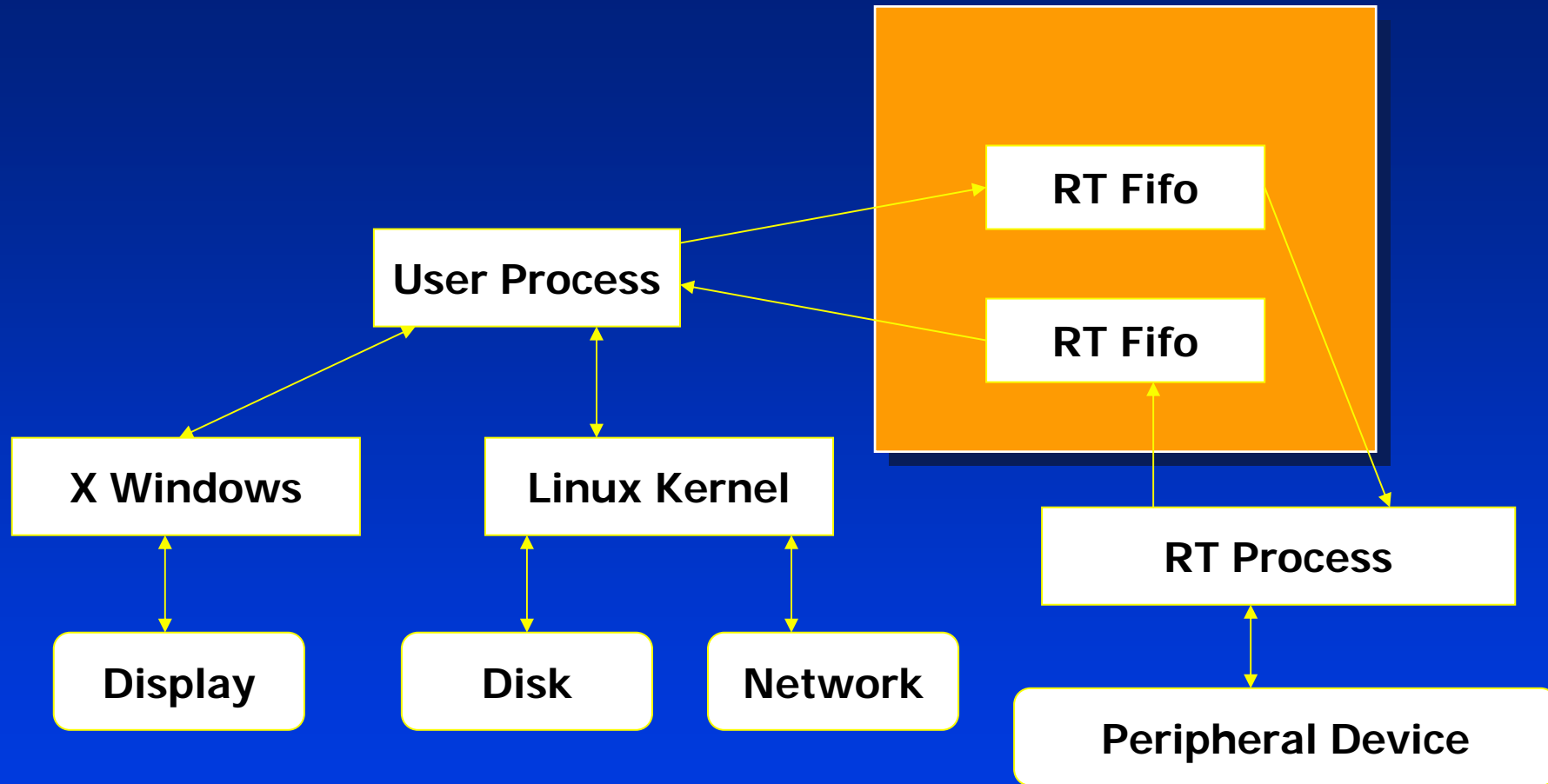
FONCTIONNALITES DE RTLinux

- Mise en service sous forme de modules Linux.
- Les tâches TR sont chargées comme des modules Linux.
- Linux apparaît comme la tâche de fond.
- RTLinux propose une API simple pour une programmation TR. L'API POSIX thread est supportée pour l'écriture de tâches TR.
- **Les tâches TR sont périodiques. Il n'y a pas de support de tâches apériodiques !**
- Les communications entre processus Linux et les tâches TR se font par des FIFOS.
- La dernière version stable supporte les processeurs x86, PowerPC et Alpha.

DEVELOPPEMENT SOUS RTLinux

- Le développement d'applications sous RTLinux suit les règles suivantes :
 - Découpage en 2 parties : TR et non TR.
 - La partie TR doit être la plus simple et la plus courte possible.
 - Le reste est non TR et sera développé dans l'espace user sous forme de processus Linux.
 - Les processus Linux et les tâches TR pourront communiquer par des FIFOs ou par mémoire partagée.
- L'API POSIX thread existant déjà sous Linux a été portée sous RTLinux. Cela facilite la migration d'un développeur Linux vers RTLinux.

DEVELOPPEMENT SOUS RTLinux

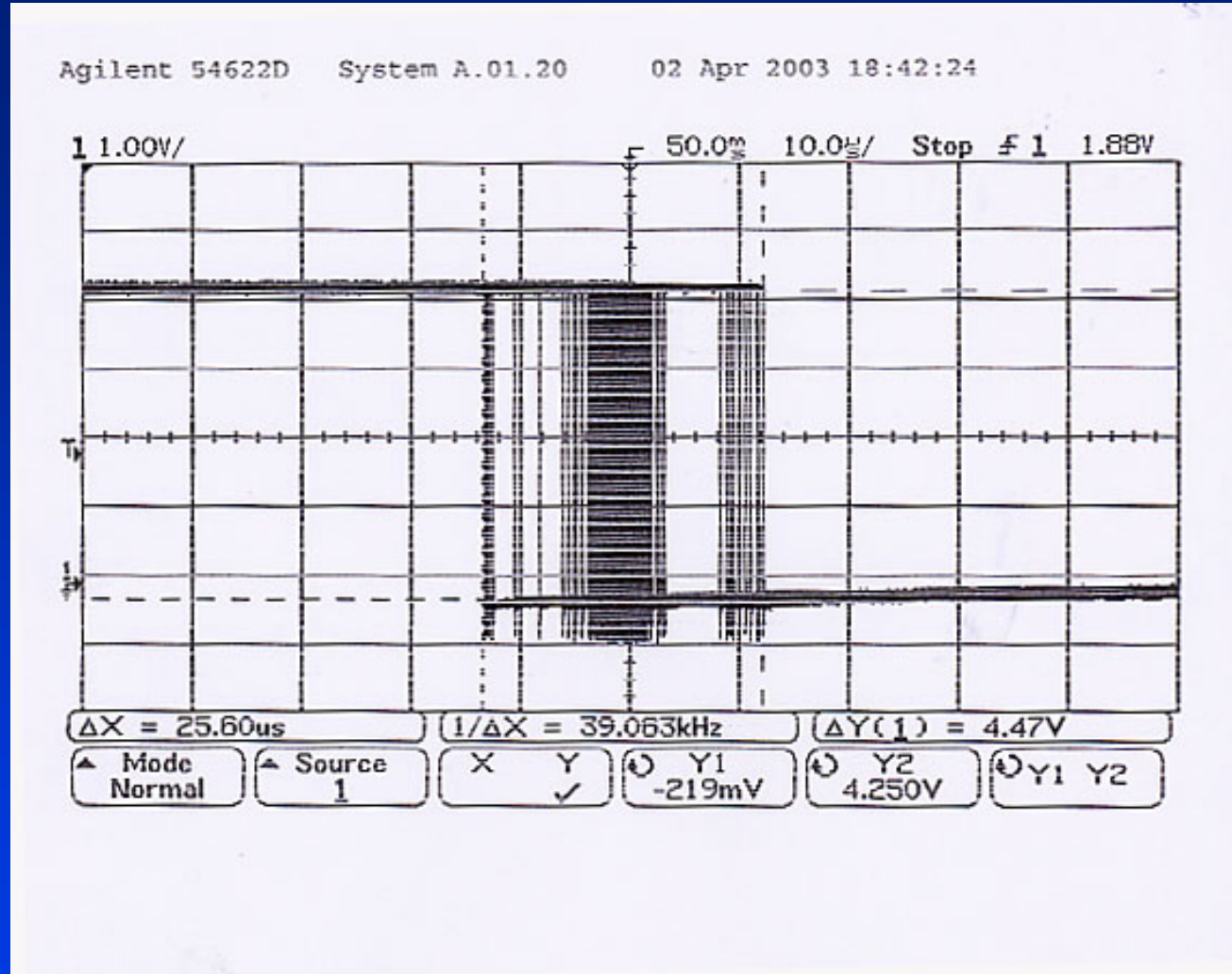


LA PETITE EXPERIENCE. SUITE ET FIN

- Mesure effectuée à l'oscilloscope lors de l'utilisation du programme rtsquare décrit précédemment modifié pour s'exécuter comme tâche Temps Réel sous RTLinux.

Dans les conditions de stress du système (écriture continue d'un fichier de 50 Mo), on obtient le résultat ci-dessous à l'oscilloscope. La mesure du jitter donne la valeur de 25.6 μ s comparés aux 17.6 ms du noyau LINUX standard, ce qui correspond environ à un rapport 1000.

LA PETITE EXPERIENCE. SUITE ET FIN



Exécution de `rtquare` avec RTLinux

LA PETITE EXPERIENCE. SUITE ET FIN

```
#include <rtl_time.h>
#include <rtl_sched.h>
#include <asm/io.h>

/* Adresse du port parallèle */
#define LPT 0x378

/* Période de sollicitation de 50 ms */
int period=50000000;

/* Valeur envoyée sur le port parallèle */
int nibl=0x01;

/* Identifiant du thread applicatif */
pthread_t thread;
```

LA PETITE EXPERIENCE. SUITE ET FIN

```
/* Corps du thread applicatif */  
void * bit_toggle(void *t)  
{  
    /* On programme un réveil de la tâche toute les 50 ms */  
    pthread_make_periodic_np(thread, gethrtime(), period);  
  
    while (1){  
        /* Ecriture de la valeur sur le port // */  
        outb(nibl,LPT);  
  
        /* Calcul de la valeur suivante (négation) */  
        nibl = ~nibl;  
  
        /* Attente du réveil */  
        pthread_wait_np();  
    }  
}
```

LA PETITE EXPERIENCE. SUITE ET FIN

```
int init_module(void)
{
    pthread_attr_t attr;
    struct sched_param sched_param;

    /* Priorité à 1 */
    pthread_attr_init (&attr);
    sched_param.sched_priority = 1;
    pthread_attr_setschedparam (&attr, &sched_param);

    /* Création du thread applicatif */
    pthread_create (&thread, &attr, bit_toggle, (void *)0);

    return 0;
}
```

LA PETITE EXPERIENCE. SUITE ET FIN

```
void cleanup_module(void)
{
    pthread_delete_np (thread);
}
```

Programme TR rtsquare pour RTLinux (fichier C rtsquare.c)

PARTIE 6 : BILAN

LE CHOIX D 'UN LINUX EMBARQUE TEMPS REEL

- Le choix est à faire en fonction des contraintes Temps Réel que doit respecter le système :
 - Pas de contrainte. Best effort. Réactivité de qq 10ms à qq 100 ms.
 - Temps Réel mou. Réactivité de qq 100 μ s.
 - Temps Réel dur. Réactivité de qq 10 μ s

LE CHOIX D 'UN LINUX EMBARQUE

TEMPS REEL

Non Temps Réel

Réactivité

Linux standard 2.6

qq 10-100 ms

PREEMPT-RT

Temps Réel mou

qq 10-100 μ s

Montavista

TimeSys

Temps Réel dur

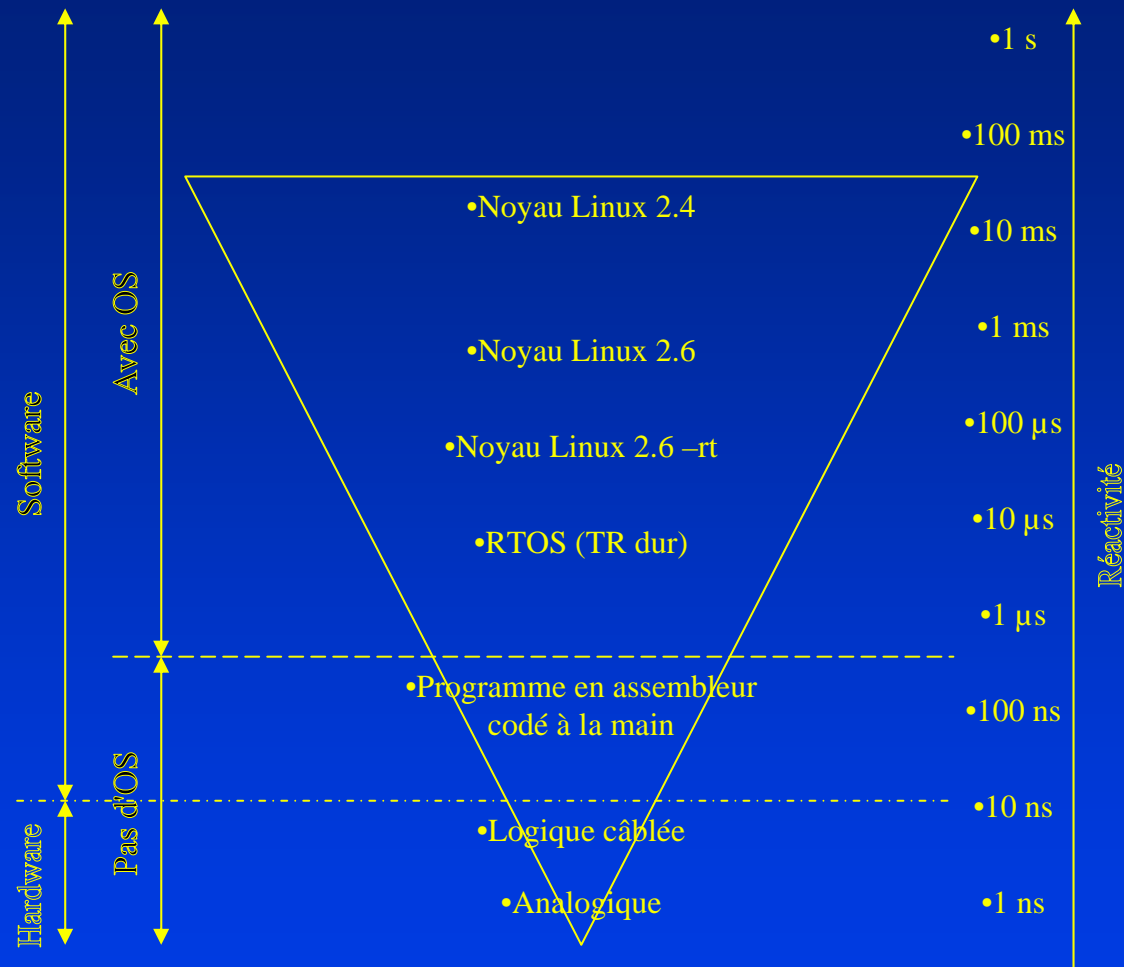
qq 10 μ s

BlueCat RT

Montavista

Xenomai, RTAI

LE CHOIX D 'UN LINUX EMBARQUE TEMPS REEL



LE CHOIX D 'UN LINUX EMBARQUE TEMPS REEL

- LINUX n'est pas fondamentalement un système d'exploitation temps réel car trop généraliste.
- Par application de patchs (PREEMPT-RT), il est possible d'avoir un système LINUX temps réel mou.
- Il est possible d'avoir un système LINUX temps dur en utilisant les extensions temps réel Xenomai, RTAI, Montavista...

LE CHOIX D 'UN LINUX EMBARQUE TEMPS REEL

- Le choix final se fera en fonction des contraintes temporelles imposées par le processus à contrôler depuis LINUX en prenant aussi en compte la complexité de mise en oeuvre dans chaque cas :
 - Configuration du noyau.
 - Ecriture de l'application temps réel.
- Choisir là aussi un linux embarqué Temps Réel commercial est rassurant. Cela a aussi un coût.

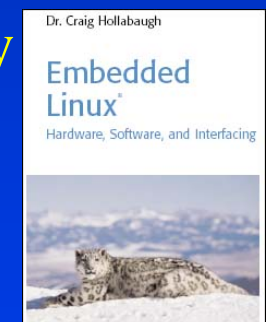
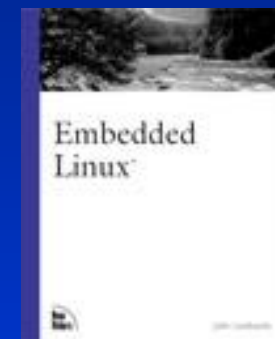
LE CHOIX D 'UN LINUX EMBARQUE

TEMPS REEL

	SIMPLE	Complexité de programmation
Linux standard 2.6		API Linux standard
PREEMPT-RT Montavista		Application de patch au noyau Linux standard API Linux standard
TimeSys		
BlueCat RT		Application de patch au noyau Linux standard Installation de modules Linux spécifiques API spécifique ou POSIX
Xenomai, RTAI	COMPLEXE	

REFERENCES BIBLIOGRAPHIQUES

- **Linux embarqué. P. Ficheux. Editions Eyrolles.
LA REFERENCE !**
- Embedded Linux. J. Lombardo. Editions New Riders.
- Embedded Linux. C. Hollabaugh. Editions Addison Wesley



REFERENCES BIBLIOGRAPHIQUES

- Embedded Systems. Firmware Demystified. E. Sutter.
Editions CMP Books
- The Art of Designing embedded Systems. J. Ganssle.
Editions Butterworth-Heinemann
- Embedded Systems Design. A S. Berger.
Editions CMP Books

