

# Modélisation Transactionnelle des Systèmes sur Puces en SystemC Ensimag 3A — filière SLE Grenoble-INP Introduction du cours

Matthieu Moy  
(transparents originaux : Jérôme Cornet)

Matthieu.Moy@imag.fr

2014-2015



## Objectifs et place du cours dans SLE

### Cours lié

- Conception et exploration d'architectures, multi-cœurs, réseaux sur puces (F. Pétrot, S. Mancini)

### Objectifs

- Une vue sur le haut du flot de conception
- Différents niveaux d'abstractions
- Exemple concret : **modélisation transactionnelle** (TLM)
- Pratique sur un outil utilisé par les industriels : SystemC
- Objectif détourné : culture générale sur les SoCs, révisions de concepts connus (cross-compilation, logiciel embarqué)

## Organisation concrète

- EnsiWiki :  
<http://ensiwiki.ensimag.fr/index.php/TLM>
- Supports de cours sur GitHub :  
<http://github.com/moy/cours-tlm>
  - ▶ `git clone` (une fois, en début de cours)
  - ▶ `git pull` (régulièrement)
  - ▶ Possibilité de travailler à plusieurs sur les squelettes de code (cf. EnsiWiki)
- Contenu de l'archive Git :
  - ▶ Transparents (\*.pdf)
  - ▶ Exemples de code (code/\*/\*.cpp), à regarder en complément du cours
  - ▶ Squelettes de code pour les TPs (TPs/)

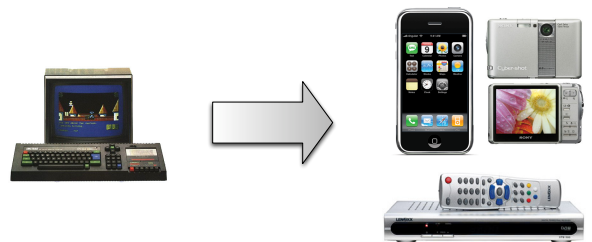
## Planning approximatif des séances

- 1 Introduction : les systèmes sur puce
- 2 Introduction : modélisation au niveau transactionnel (TLM)
- 3 Introduction au C++
- 4 Présentation de SystemC, éléments de base
- 5 Communications haut-niveau en SystemC
- 6 Modélisation TLM en SystemC
- 7 TP1 : Première plateforme SystemC/TLM
- 8 Utilisations des plateformes TLM
- 9 TP2 (1/2) : Utilisation de modules existants (affichage)
- 10 TP2 (2/2) : Utilisation de modules existants (affichage)
- 11 Notions Avancé en SystemC/TLM
- 12 TP3 (1/3) : Intégration du logiciel embarqué
- 13 TP3 (2/3) : Intégration du logiciel embarqué
- 14 TP3 (3/3) : Intégration du logiciel embarqué
- 15 05/01 : Intervenant extérieur : Jérôme Cornet
- 16 Perspectives et conclusion

## Sommaire

- 1 Systèmes sur Puce (SoC)
- 2 Modélisation au niveau transactionnel

## Évolution des besoins du grand public



## Quelques exemples

- Lecteurs MP3



### Besoins techniques

- Mini ordinateur (interface utilisateur)
- Codecs (décodage/encodage) audio (MP3, etc.)
- Pilotage de disque dur
- Périphériques (USB, IEEE 1394...)
- Autonomie

## Quelques exemples

- Téléphones portables



### Besoins techniques

- Mini ordinateur (interface utilisateur, applications embarquées)
- Traitement de Signal (technologie de transmission)
- Périphériques (USB, Capteur CCD)
- Autonomie

# Quelques exemples

- Télévision numérique
  - ▶ Lecteurs/Enregistreurs DVD
  - ▶ Démodulateurs Satellite, Décodeurs TNT (*set-top boxes*)
  - ▶ Télévision Haute-définition (HD-TV, UHD...)

## Besoins techniques

- ✱ Mini ordinateur (interface utilisateur)
- ✱ Pilotage de disque dur, lecteur/graveur DVD
- ✱ Fonctions graphiques (compositions, zoom, curseur...)
- ✱ Encodage/décodage vidéo (MPEG2, MPEG2 HD, MPEG 4, H264, HEVC...)
- ✱ Encodage/décodage audio (PCM, AC3, AAC, MP3...)
- ✱ Périphériques (IEEE 1394, S/PDIF, HDMI...)

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 10 / 80 >

# Évolutions techniques

## Microprocessor Transistor Counts 1971-2011 & Moore's Law

Transistor count

curve shows transistor count doubling every two years

Date of introduction

Source : Wikipedia

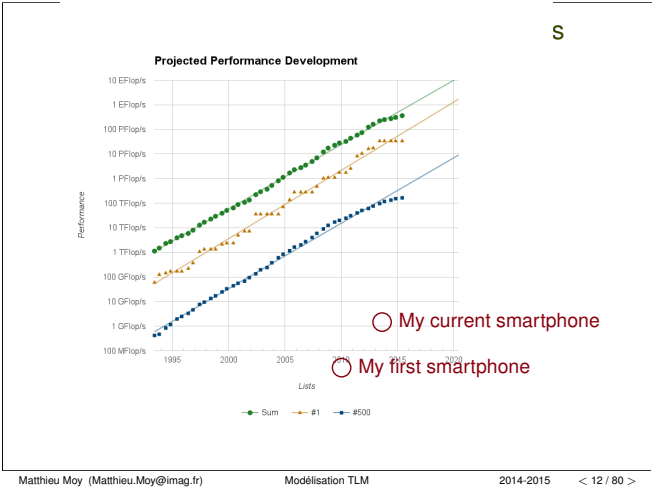
([http://en.wikipedia.org/wiki/File:Transistor\\_Count\\_and\\_Moore%27s\\_Law\\_-\\_2011.svg](http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg))

Processor	Transistor Count	Year
4004	2300	1971
8008	6000	1972
8080	29000	1974
8085	60000	1976
8088	290000	1982
80286	275000	1982
80386	275000	1985
80486	1200000	1989
586	2750000	1993
Pentium	3100000	1995
Pentium II	7500000	1997
Pentium III	9500000	1999
Celeron	9500000	2000
Itanium	20000000	2001
Itanium 2	20000000	2002
Itanium 2M	20000000	2003
Itanium 2E	20000000	2004
Itanium 2P	20000000	2005
Itanium 2N	20000000	2006
Itanium 2M	20000000	2007
Itanium 2P	20000000	2008
Itanium 2N	20000000	2009
Itanium 2M	20000000	2010
Itanium 2P	20000000	2011

Source : Wikipedia

([http://en.wikipedia.org/wiki/File:Transistor\\_Count\\_and\\_Moore%27s\\_Law\\_-\\_2011.svg](http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg))

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 11 / 80 >




# Évolutions techniques

- Évolution de la complexité des fonctions réalisables



Portes NAND



Contrôleur USB



Chipset



System-on-Chip

Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2014-2015

< 13 / 80 >

# Définition

**System-on-Chip (Système sur puce)**

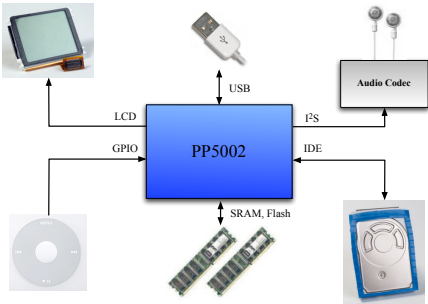
*Puce regroupant tous les éléments électroniques (micro-processeur, composants spécifiques...) nécessaires à la réalisation d'un système (produit) complet.*

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 14 / 80 >

# Quelques utilisations des SoC

# iPod

- SoC : PP5002 (Portal Player)



The diagram illustrates the PP5002 SoC (Portal Player) as a central component connected to various peripherals:

- LCD**: Connected to the top of the PP5002 SoC.
- USB**: Connected to the top of the PP5002 SoC.
- Audio Codec**: Connected to the right of the PP5002 SoC, with earphones shown as an output.
- GPIO**: Connected to the left of the PP5002 SoC, with a CD/DVD disc shown as an input/output device.
- SRAM, Flash**: Connected to the bottom of the PP5002 SoC.
- IDE**: Connected to the bottom right of the PP5002 SoC, with a hard drive shown as an output device.

Matthieu Moy (Matthieu.Moy@imag.fr)

Modélisation TLM

2014-2015 < 16 / 80 >

# Décodeur Satellite CDVB2300B

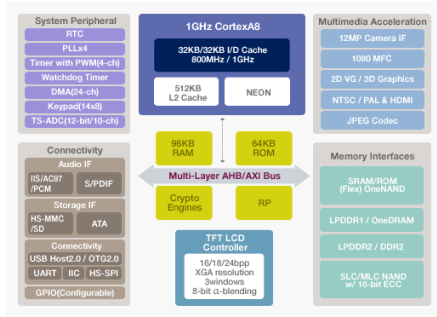
- SoC : STi5518 (STMicroelectronics)

The diagram illustrates the system architecture centered around the STi5518 SoC. Key components and their connections include:

- Inputs:** A satellite dish and a remote control provide input to the STi5518 SoC. Two front-end modules, STV0299 QPSK and STV0360 COFDM, also feed into the SoC.
- Storage and Memory:** The SoC is connected to SDRAM, an IDE interface with a Flash drive and an HDD, and a Modem.
- Outputs:** The SoC outputs to an Overdrive connector, a television (supporting PAL and NTSC Digital), and an Analog/S/PDIF interface (labeled 5.1).

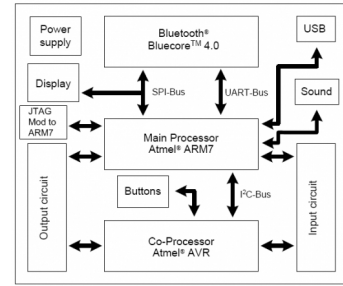
Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 17 / 80 >

## S5PC110 (Samsung Galaxy S Smartphone)

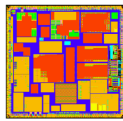


Source : [http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly\\_id=834&partnum=S5PC110](http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=834&partnum=S5PC110)

## Brique lego NXT



Source : <http://iar.com/website1/1.0.1.0/1518/1/>



## Composants des SoC (non exhaustif)

## Microprocesseurs

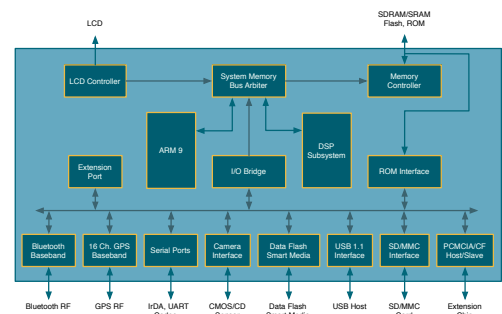
- CPU (Central Processing Unit) : processeur généraux
  - Fonctions : contrôle, interface utilisateur, traitements légers
  - Exemples : ARM9, ST20, SH4, Leon mais pas Intel core i7!...
- DSP (Digital Signal Processor)
  - Fonctions : Traitement du signal, calculs complexes
  - Exemples : Ti TMS320C55x, etc.
- Processeurs VLIW (Very Long Instruction Word)
  - Fonctions : traitement multimédia
  - Exemples : ST210, Kalray...
- Supports de la partie logicielle du système

## Mémoires

- ROM, RAM, Flash...
  - Souvent hors du SoC
- Dans la puce :
  - Contrôleur(s) mémoires
  - Petites mémoires internes
  - Mémoires caches, « fifo » (tampons/files d'attente)
- Fonctions
  - Stockage temporaire (RAM)
  - Programme interne (ROM), possibilité de mise à jour (Flash)
  - Stockage (Flash)

## Exemple : PDA GPS

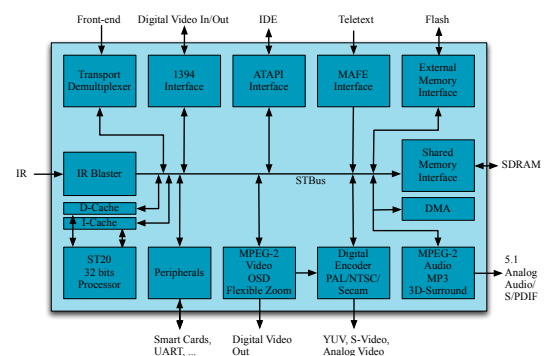
- SoC : Atlas-M (Centrality)



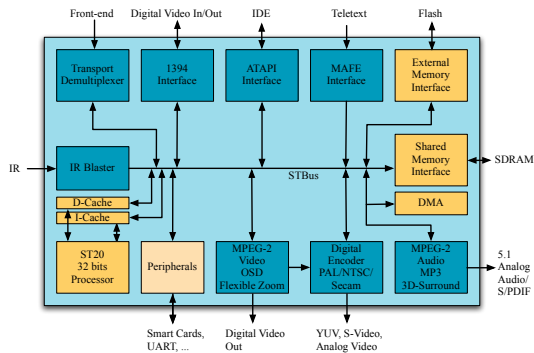
## Composants utilitaires

- DMA (Direct Memory Access)
  - Transferts mémoires/mémoires, mémoires/périphériques
  - Décharge le CPU (pas d'attente liée aux transferts)
- Timer, RTC (Real-Time Clock)
  - Mesure de l'écoulement du temps
  - Utilisations :
    - ★ Contrôle du nb d'images par secondes
    - ★ Programmation de délais d'expiration
    - ★ Utilisation par OS Temps Réel
- Contrôleur d'interruptions (ITC)
  - Centralisation de tous les signaux d'interruptions
  - Informations sur l'émetteur de l'interruption

## Exemple : STI5518



## Exemple : STi5518 - CPU, Mémoires, Utilitaires



## Entrées/Sorties (1/2)

- GPIO (General Purpose Inputs/Outputs)
  - programmation/lectures de broches du circuit
  - Utilisation : lectures de boutons, clavier simple
- Composants ports série
  - UART (Universal Asynchronous Receiver/Transmitter) : port RS232
  - SSP (Synchronous Serial Port) : port série haute vitesse
  - Utilisation : branchement de composants externes, debugage
- Contrôleurs LCD (numérique)

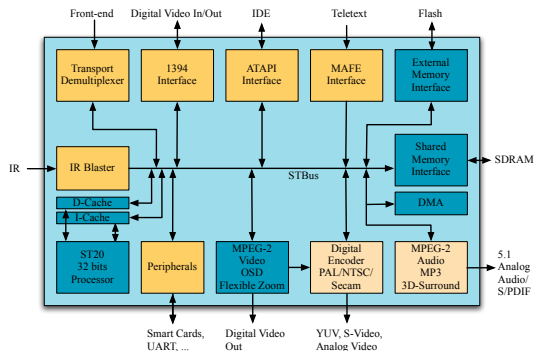
## Entrées/Sorties (2/2)

- Pilotage de bus de périphériques
  - IDE/ATA...
    - ★ Disque dur interne
    - ★ DVD
  - USB
    - ★ Host : branchement de périphériques externes
    - ★ Peripheral : branchement à un ordinateur
  - IEEE 1394 aka FireWire (caméscopes numériques, ...)
    - ★ Caméscopes numériques
    - ★ Liaisons haute vitesse
- Composants RF
  - Bluetooth
  - GPS
  - GSM...

## Parties analogiques

- Générations d'horloges, reset...
- ADC, DAC (Analog/Digital Converter) : entrées/sorties analogiques
  - Audio
  - Vidéo
  - Commandes d'actionneurs
  - Capteurs

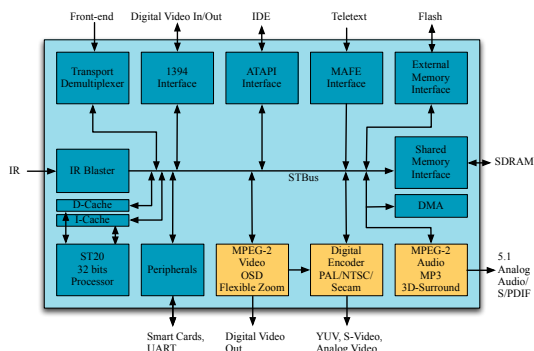
## Exemple : STi5518 - E/S, Analogique



## Composants dédiés

- Accélération d'une fonction particulière
  - Serait trop lent en logiciel !
  - Compromis
    - ★ Complexité
    - ★ Accélération
    - ★ Consommation
    - ★ Flexibilité
- Plusieurs possibilités de réalisation :
  - 100% matérielle
    - ★ ex : Décodeurs Vidéos « câblés »
  - Semi-matérielle
    - ★ Utilisation d'un cœur de processeur pour simplifier le matériel
    - ★ ex : Décodeurs audios, traitements plus difficiles à câbler

## Exemple : STi5518 - Composants dédiés



## Réseau d'interconnection (interconnect)

- Principal moyen de communication entre composants
- Bus partagés
  - Communications par plages mémoires
  - Arbitrage des communications
  - Un peu analogue aux bus « PC » actuels mais
    - ★ Pas de connectique
    - ★ Basse consommation
  - Paramètres : vitesse, largeur...
  - exemple : AmbaBus (ARM) AHB, APB, AXI...
- Networks-on-Chip (NoC)
  - Réseau complexe construit en fonction des besoins du SoC
  - exemple : STBus

# Exemple détaillé (Sharp/NXP LH7-A400 SoC)



# Le logiciel (embarqué)

- Plusieurs logiciels sur architectures différentes
  - ▶ Mélange SH4, ARM, ST...
  - ▶ Endianess différentes
  - ▶ Problèmes techniques d'ordre de boot
  - ▶ Problèmes de synchronisations
- Communications de base : accès au bus et interruptions
- Partie génie logiciel
  - ▶ Utilisation d'OS multitâche/temps réel : Linux, OS/20, Windows CE...
  - ▶ Factorisation du code bas niveau dans des pilotes (drivers)
  - ▶ Couche supplémentaire au dessus de l'OS : le middleware

---

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 35 / 80 >

- 
- Matthieu Moy (Matthieu.Moy@imag.fr)
Modélisation TLM
2014-2015
< 35 / 80 >

# Exemple de structure logicielle (set-top-box)

The diagram illustrates a software architecture stack for a set-top box, organized into several layers and components:

- Top Layer (Applications):** Consists of four separate blocks: EPG, E-Mail, Web Browser, and Other Apps.
- Middleware Layer:** A single block labeled "Middleware (MediaHighway, OpenTV, Liberate,...)" that sits below the applications.
- Adaption Layer:** A single block labeled "Adaption Layer" that sits below the middleware.
- OS/20 RTOS Layer:** Consists of two blocks: "OS/20 RTOS" and "STAPI Drivers (audio, video, OSD, tuner, smartcard,...)".
- Bottom Layer (Hardware):** A single block labeled "OMEGA Hardware (eg. STI5518)".

Arrows indicate the flow of data and control between these layers, showing a clear separation of concerns and a layered architecture.

Source : doc STI5518

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 36 / 80 >



# Exemple de structure logicielle (Android)

The diagram illustrates the Android system architecture as a stack of layers:

- APPLICATIONS**: Home, Contacts, Phone, Browser, ...
- APPLICATION FRAMEWORK**: Activity Manager, Window Manager, Content Providers, View System, Notification Manager, Package Manager, Telephony Manager, Resource Manager, Location Manager, XMPP Service.
- LIBRARIES**: Surface Framework, Media Framework, SQLite, OpenGL ES, FreeType, WebKit, SQL, SSL, libc.
- ANDROID RUNTIME**: Core Libraries, Dalvik Virtual Machine.
- LINUX KERNEL**: Display Driver, Camera Driver, Bluetooth Driver, Flash Memory Driver, Binder (IPC) Driver, USB Driver, Keypad Driver, Wifi Driver, Audio Drivers, Power Management.

**Source** : <http://en.wikipedia.org/wiki/File:Android-System-Architecture.svg>

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 37 / 80 >



---

Matthieu Moy (Matthieu.Moy@imag.fr)
Modélisation TLM
2014-2015
< 38 / 80 >

# Complexité croissante de conception

- Nombre de transistors : + 50% par an (Moore)
- Productivité en conception : + 30% par an  
⇒ « **Design Gap** »
- Besoin incessant de nouvelles techniques de conception

---

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 39 / 80 >

- 
- Matthieu Moy (Matthieu.Moy@imag.fr)
Modélisation TLM
2014-2015
< 39 / 80 >

# Complexité croissante de conception

- Circuits à assembler énormes
  - ▶ Multiples éléments complexes : processeurs, réseau d'interconnection...
  - ▶ Parfois plusieurs mini-SoC dans le SoC (sous-ensembles simples CPU+DMA+Mémoire)
  - ▶ Évolution possible vers le massivement multiprocesseur
- VHDL/Verilog suffisent difficilement...
- Pas de révolution type « niveau porte → RTL » pour l'instant

---

Matthieu Moy (Matthieu.Moy@imag.fr)      Modélisation TLM      2014-2015      < 40 / 80 >

- 
- |                                     |                  |           |             |
|-------------------------------------|------------------|-----------|-------------|
| Matthieu Moy (Matthieu.Moy@imag.fr) | Modélisation TLM | 2014-2015 | < 40 / 80 > |
|-------------------------------------|------------------|-----------|-------------|

# Hardware/Software Design Flow

The diagram illustrates two design flows over time. The Traditional Design-Flow (left) is a linear sequence: Specification, Algorithm → RTL Design → Synthesis → Factory → Software Development → Integration → Validation. The Model based Design-Flow (right) is iterative: Specification, Algorithm → RTL Design → Synthesis → Factory → Validation. It includes a TLM Model and Software Development (both in red boxes) that interact with the RTL Design and Synthesis stages. A red arrow labeled 'cost > 1,000,000 \$!' points from the Synthesis stage of the traditional flow to the Factory stage of the model-based flow. A red arrow labeled 'gain' points from the Validation stage of the model-based flow back to the Factory stage. A vertical arrow on the left indicates the progression of time.

```
graph TD
    subgraph Traditional_Design-Flow [Traditional Design-Flow]
        direction TB
        T1[Specification, Algorithm] --> T2[RTL Design]
        T2 --> T3[Synthesis]
        T3 --> T4[Factory]
        T4 --> T5[Software Development]
        T5 --> T6[Integration]
        T6 --> T7[Validation]
    end

    subgraph Model_based [Model based]
        direction TB
        M1[Specification, Algorithm] --> M2[RTL Design]
        M2 --> M3[Synthesis]
        M3 --> M4[Factory]
        M4 --> M5[Validation]
        M1 --> M2
        M2 --> TLM[TLM Model]
        TLM --> M3
        M3 --> SD[Software Development]
        SD --> M4
        M4 -.-> TLM
        M4 -.-> SD
    end

    T3 -- "cost > 1,000,000 $!" --> M4
    M5 -- "gain" --> M4
```

Time

Traditional Design-Flow

Model based

Specification, Algorithm

RTL Design

Synthesis

Factory

Software Development

Integration

Validation

TLM Model

Software Development

Integration

Validation

cost > 1,000,000 \$!

gain

41 / 80



## Durée du cycle de développement

- Évolution rapide
- Dates à ne pas manquer (Noël, nouvel an chinois, Consumer Electronics Show...)
- ⇒ Un produit prêt 6 mois trop tard est difficilement vendable !
- ⇒ Le « Time to market » est aussi important que la main d'œuvre totale.

## Coût d'une erreur

### Bug dans le logiciel

- Mise à jour du firmware
- Pas forcément acceptable partout... (difficulté pour l'utilisateur, systèmes critiques)

### Bug dans le matériel

- Fabrication de nouveaux masques
- Exemple de coût :

Finesse de gravure	0.25 $\mu m$	0.13 $\mu m$	65 nm
Coût masque 1 couche	10 000 \$	30 000 \$	75 000 \$
Nb de couches	12	25	40
Coût total	120 000 \$	750 000 \$	3 M\$

source EETimes

## Coût d'une erreur

### Bug dans le matériel (suite)

- Circuit déjà fabriqué : recherche d'un contournement (workaround)
- Valable en technologie ASIC
- SoC FPGA
  - ▶ ARM Excalibur : ARM 922 (200 MHz) + FPGA APEX 20KE
  - ▶ Xilinx Virtex 4 : PowerPC 405 (450 MHz) + FPGA + Ethernet MAC
  - ▶ Là encore, mise à jour limitée

## Problèmes de conception

- Vitesse de simulation
  - ▶ Simulation du SoC niveau RTL : plusieurs heures, voire jours...
  - ▶ ex : Encodage et décodage d'une image en MPEG 4 = 1 h en simulation RTL
  - ▶ Impossibilité de tester le(s) logiciel(s) embarqué(s) à ce niveau
  - ▶ Moins de temps disponible pour valider le système...
  - ▶ Développement séparé des différents blocs
  - ▶ Quelques solutions couramment pratiquées :
    - ★ Cosimulation
    - ★ Émulation matérielle

## Problèmes d'intégration

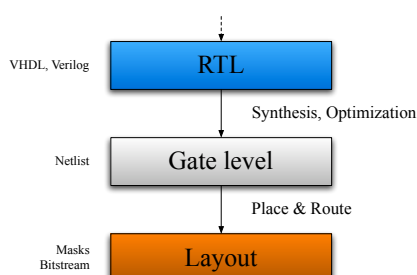
- Fonctionnelle
  - ▶ Développement séparé des composants, réutilisation
  - ▶ Aucune garantie de fonctionnement
  - ▶ Problèmes de compatibilité plus complexes qu'électroniques
- Performances
  - ▶ Adéquation d'un ensemble de composants pour réaliser une tâche dans un temps donné
  - ▶ Dépendances non fonctionnelles complexes

## Problèmes de validation

- Conformité du système à la spécification ?
- Spécification de plus en plus complexes
  - ▶ Normes MPEG x, H264, HEVC ...
  - ▶ Formats informatiques divers
  - ▶ Interprétation parfois erronée
- Volumes de données de plus en plus gros (vidéo HD, FHD, 4K, bientôt 8K)
- Format de la spécification ?

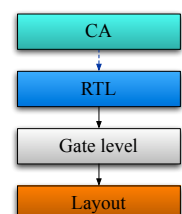
## Retour sur le flot de conception

- Bas du flot de conception (vu en 2A)
  - 3 principaux niveaux d'abstraction

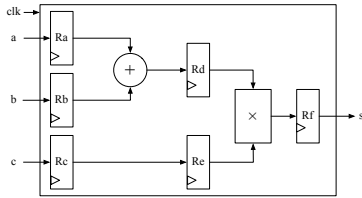


## Niveau supérieur : Cycle Accurate (1/2)

- Objectif : simulation rapide
- Caractéristiques
  - ▶ Précis au cycle d'horloge près
  - ▶ Précis au niveau données (bit true)
  - ▶ Écriture libre du modèle interne des composants (C, C++...)



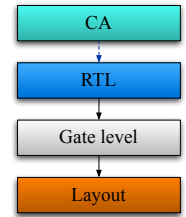
## Cycle Accurate : exemple



```
void my_function() // pseudo-code tres approximatif ...
{
    Ra = a;
    Rb = b;
    Rc = c;
    wait(clk, 2);
    s = (Ra+Rb)*Rc;
}
```

## Niveau supérieur : Cycle Accurate (2/2)

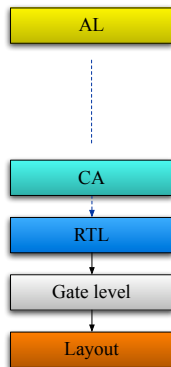
- Utilisation
  - Modèles de remplacement pour simulations RTL
  - Ex : modèles de processeurs...



- Inconvénients
  - Non synthétisable (référence = RTL)
  - Effort de modélisation important

## Niveau « Algorithmique »

- Programme décrivant la fonctionnalité (C, Matlab, etc.)
- Souvent séquentiel
- Ex : algorithme de référence décodage MPEG
- Écriture non ambiguë d'une portion de la spécification
- Pas de référence au matériel :
  - Pas de partitionnement hard/soft
  - Partie contrôle absente



## Exemple : décodage XVID

```
...
bound = 0;
for (y = 0; y < mb_height; y++) {
    cp_mb = st_mb = 0;
    for (x = 0; x < mb_width; x++) {
        MACROBLOCK *mb;

        /* skip stuffing */
        while (BitstreamShowBits(bs, 10) == 1)
            BitstreamSkip(bs, 10);

        if (check_resync_marker(bs, fcode - 1)) {
            bound = read_video_packet_header(bs, dec, fcode - 1,
                &quant, &fcode, NULL, &intra_dc_threshold);
            x = bound % mb_width;
            y = bound / mb_width;
        }
        mb = &dec->mbs[y * dec->mb_width + x];

        DPRINTF(XVID_DEBUG_MB, "macroblock (%i,%i) %08x\n", x, y, BitstreamShowBits(bs, 32));

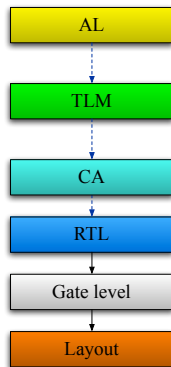
        if (!!(BitstreamGetBit(bs))) { /* block _is_ coded */
            uint32_t mcbpc, cbpc, cbpy, cbp;
            uint32_t intra, acpred_flag = 0;
            int mcsel = 0; /* mcsel: '0'-local motion, '1'-GMC */

            cp_mb++;
            mcbpc = get_mcbpc_inter(bs);
            mb->mode = mcbpc & 7;
            cbpc = (mcbpc >> 4);

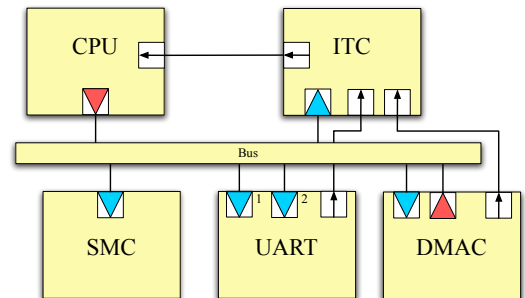
            DPRINTF(XVID_DEBUG_MB, "mode %i\n", mb->mode);
            DPRINTF(XVID_DEBUG_MB, "cbpc %i\n", cbpc);
        }
    }
}
```

## Niveau transactionnel : TLM

- Transaction Level Modeling
- Plusieurs définitions...  
ici : TLM IEEE 1666-2011 (standard)
- Objectif n°1 : développement du logiciel embarqué
- Caractéristiques
  - Pas d'horloge (cycle-less)
  - Bit true
  - Définition, modélisation précise des communications



## Un exemple



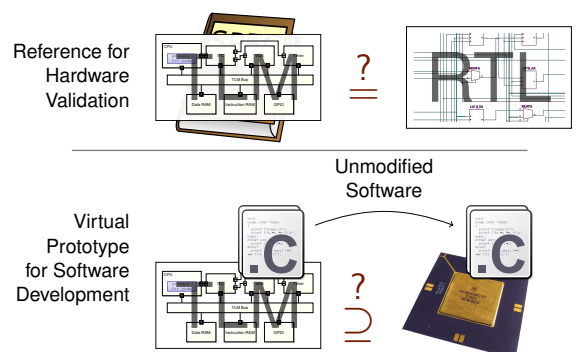
## Objectif développement du logiciel embarqué

### Contrat entre logiciel embarqué et matériel

- Logiciel correct en simulation TLM  $\Rightarrow$  logiciel correct en simulation RTL et sur puce
- Nécessite tous les éléments matériels « visibles » par le logiciel

$\Rightarrow$  Modéliser tout ce qui est nécessaire au logiciel embarqué, mais seulement cela

## Uses of Functional Models



## Éléments nécessaires

- Possibilité d'actions lectures/écritures
- Plages d'adresses
- Mémoires
- Bancs de registres
- Interruptions

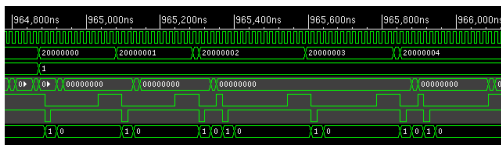
## Idée principale

- Actions de lectures/écritures = **transactions** sur le réseau d'interconnexion
- Correspondance avec les opérations logiques effectuées sur le bus

### Informations minimales contenues dans une transaction

- Type : lecture ou écriture
- Adresse de base
- Données
- Taille (nb de données)
- État : OK, Erreur, Temps dépassé...
- Implicite : taille d'une donnée élémentaire, taille d'une adresse

## Ex : lectures successives sur bus AHB



RTL

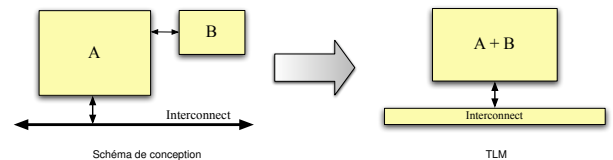


TLM

Abstraction des communications sur bus

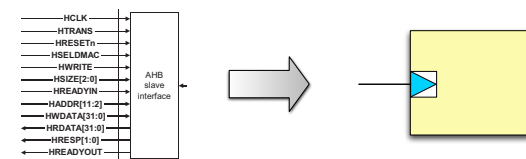
## Composants

- Séparation en fonction des communications
  - ▶ Fusion de composants communiquant en dehors du réseau d'interconnexion
  - ▶ Règle non absolue (fonction des besoins et des types de communications disponibles)
- Possibilité de hiérarchie
- Simulation indépendante de chaque composant :  
→ **concurrency**



## Ports de communication (transactions)

- Deux types de ports (de façon générale)
  - ▶ Initiateurs (**initiator**, **master**)
    - ★ « Émetteurs » des transactions
  - ▶ Cible (**target**, **slave**)
    - ★ « Répondent » aux transactions
    - ★ Partie passive d'un composant
    - ★ Association port cible ↔ plage d'adresse
- Analogie avec le réseau
  - ▶ initiateur ≈ client
  - ▶ cible ≈ serveur
  - ▶ **⚠ Rien à voir avec la distinction lecture/écriture !**
- Exemple :



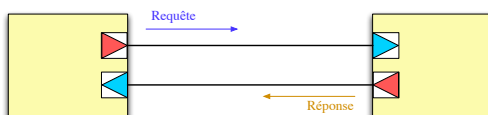
## Protocole (1/2)

- Ensemble des actions possibles au niveau des ports (lecture, écriture, ...)
- Protocoles **bloquants**



## Protocole (2/2)

- Protocoles **non bloquants**
  - ▶ Ne bloquent pas l'**initiateur**
  - ▶ Deux canaux de communication : requête et réponse

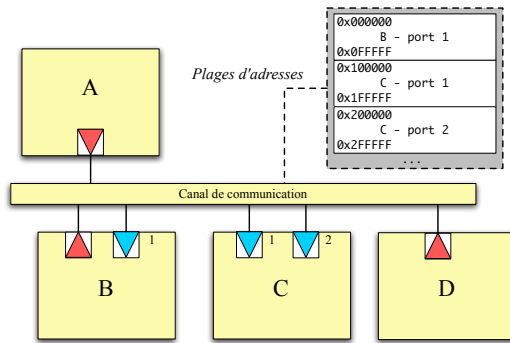


## Canal de communication (1/2)

- Abstrait les différents éléments du bus
  - ▶ Décodage
  - ▶ Multiplexage
  - ▶ Signaux...
- Fonctionnalité principale : routage des transactions
- Connaissance des plages d'adresses des composants
- Ports de communications spéciaux (cible : plusieurs-vers-un, initiateur : un-vers-plusieurs)



## Canal de communication (2/2)

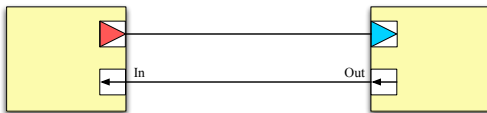


## Interruptions (1/2)

- Qu'est-ce que c'est ?
  - Connection directe entre composants par un fil
  - Unidirectionnel
- À quoi ça sert ?
  - Processeurs
    - Déclenchement immédiat de routines en cours d'exécution du code normal
    - ex : Interruption Souris, traitement terminé par un autre composant...
    - Économies d'énergies (HLT sur Pentium, ...)
  - Autres composants matériels
    - Synchronisation
    - « Arrêt d'urgence » d'un composant

## Interruptions (2/2)

- Modélisation non standardisée...
- Une solution : signaux booléens (RTL)
  - Sensibilité sur fronts (montants, descendants)
  - Entrées, sorties de signaux (idem RTL)
  - Connections point à point
  - Principal moyen de synchronisation pour les cibles



## Modélisation interne d'un composant

- Liberté de codage... mais respect du niveau d'abstraction !

### Partie « initiateur »

- Transactions générées réalistes
  - Mêmes adresses
  - Données précises

### Partie « cible »

- Mémoires
    - Respect de la taille du données
    - Endianess
  - Bancs de registres
    - Adresses relatives des différents registres
    - Actions associées
- Contrat d'utilisation du composant

## Exemple de bancs de registres (DMAC ARM)

Name	Address (base+)	Type	Reset value	Description
DMACIntStatus	0x000	RO	0x00	See Interrupt Status Register on page 3-10
DMACIntTCStatus	0x004	RO	0x00	See Interrupt Terminal Count Status Register on page 3-10
DMACIntTCClear	0x008	WO	-	See Interrupt Terminal Count Clear Register on page 3-11
DMACIntErrorStatus	0x00C	RO	0x00	See Interrupt Error Status Register on page 3-11
DMACIntErrClr	0x010	WO	-	See Interrupt Error Clear Register on page 3-12
DMACRawIntTCStatus	0x014	RO	-	See Raw Interrupt Terminal Count Status Register on page 3-13
DMACRawIntErrorStatus	0x018	RO	-	See Raw Error Interrupt Status Register on page 3-13
DMACEnbldChms	0x01C	RO	0x00	See Enabled Channel Register on page 3-14
DMACSoftBReq	0x020	R/W	0x0000	See Software Burst Request Register on page 3-14
DMACSoftSReq	0x024	R/W	0x0000	See Software Single Request Register on page 3-15

## Exemple de mode d'emploi de registre (DMAC ARM)

### 3.4.10 Software Single Request Register

The read/write DMACSoftSReq Register, with address offset of 0x024, enables DMA single requests to be generated by software. You can generate a DMA request for each source by writing a 1 to the corresponding register bit. A register bit is cleared when the transaction has completed. Writing 0 to this register has no effect. Reading the register indicates the sources that are requesting single DMA transfers. You can generate a request from either a peripheral or the software request register. Figure 3-10 shows the bit assignments for this register.

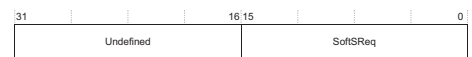


Figure 3-10 DMACSoftSReq Register bit assignments

Table 3-11 lists the bit assignments for this register.

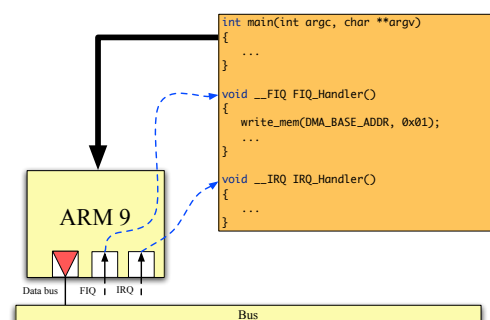
Table 3-11 DMACSoftSReq Register bit assignments

Bits	Name	Function
[31:16]	-	Read undefined. Write as zero.
[15:0]	SoftSReq	Software single request.

## Insertion du logiciel embarqué (1/2)

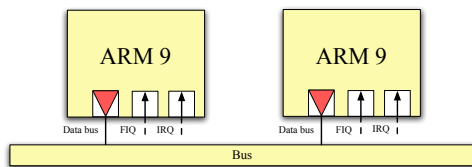
- « Emballage » autour du code (wrapper)
- Interface
  - Accès mémoire via un bus de données
    - port initiateur
  - Entrée(s) interruptions
  - Dépendant du modèle de processeur...
- Mise en correspondance interruptions ↔ code de traitement

## Insertion du logiciel embarqué (2/2)



## Question

- Communication d'une valeur entière  $x$  entre les processeurs ?



## Question



Comment faire ? Faut-il ajouter des composants ?

## Comparaison avec les autres niveaux (1/2)

### TLM vs. Algorithmique

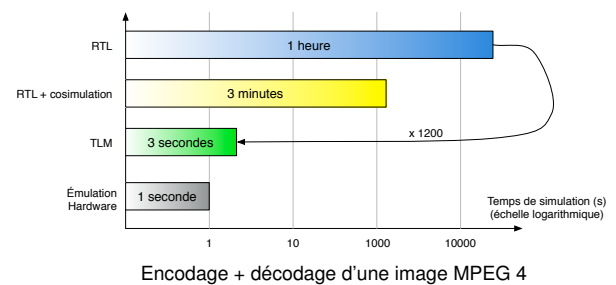
- Découpage de l'algorithme en blocs indépendants
- Validation du fonctionnement en parallèle
- Aspect composant
  - Réutilisation
  - Hiérarchie de composants
- Partitionnement

## Comparaison avec les autres niveaux (2/2)

### TLM vs. Cycle accurate

- Vitesse de simulation
  - Écriture libre de l'intérieur du composant
  - Communications abstraites
  - Comportement asynchrone
  - Dépend de l'implémentation !
- Précision des données
- Modélisation facile : réutilisation du code de niveau AL

## Vitesse de simulation



## Apports de TLM

- Écriture du code embarqué possible en avance de phase !
  - Vitesse de simulation
  - Facilité de modélisation
- Debuggage de l'intégration des composants
- Nouveau niveau de référence
  - Moyen de communications entre monde du hard et monde du soft...
  - Référence disponible en avance de phase
- Analyse d'architecture

## Synthèse

- Synthèse comportementale : peu en industrie
    - Quelques essais : MathLab/Simulink → RTL, C → RTL, ...
    - Nécessité d'un niveau d'abstraction bien défini
  - TLM
    - Précision des communications uniquement
    - Intérieur du composant : aucune règle (pointeurs, bibliothèque, etc.)
    - Synthèse éventuelle :
      - ★ Partie connection au bus
      - ★ Réseau d'interconnexion
      - ★ Portions de code avec algorithmique simple
- ⇒ en général, RTL et TLM sont écrits à la main et indépendamment.