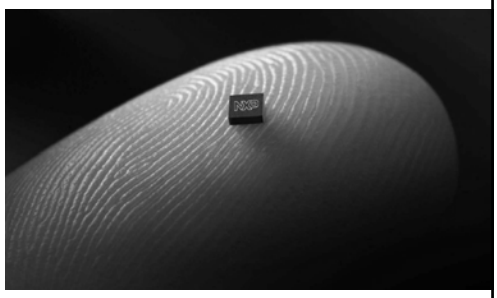
# Modelling of Systems on Chip

**Part of Tutorial A:**
**Automatically Realising Embedded Systems**
**From High-Level Functional Models**

Wido Kruijtzer, Victor Reyes
NXP Semiconductors
March 10, 2008

## Outline

- **NXP Products / challenges**
- Abstraction Levels
- Functional Modeling
- Architecture Modeling
- Summary



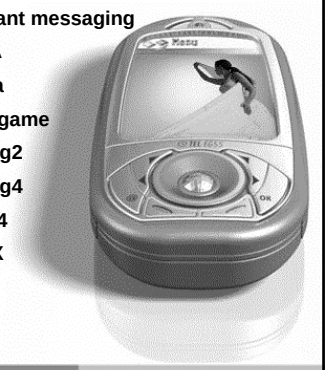
# NXP Semiconductors – Reborn and Renewed

- Spin-out of Royal Philips Electronics' Semiconductor Division
- #2 in Europe, Top-10 global supplier
- Sales of € 4.6 billion in 2007
- 37,000 employees / 7,500 engineers
- Investing € 950 million in R&D annually
- 25,000+ patents
- More than 26 R&D centers in 12 countries
- Headquarters: Eindhoven, The Netherlands
- Key focus areas:
  - Mobile & Personal, Home, Automotive & Identification, Multimarket

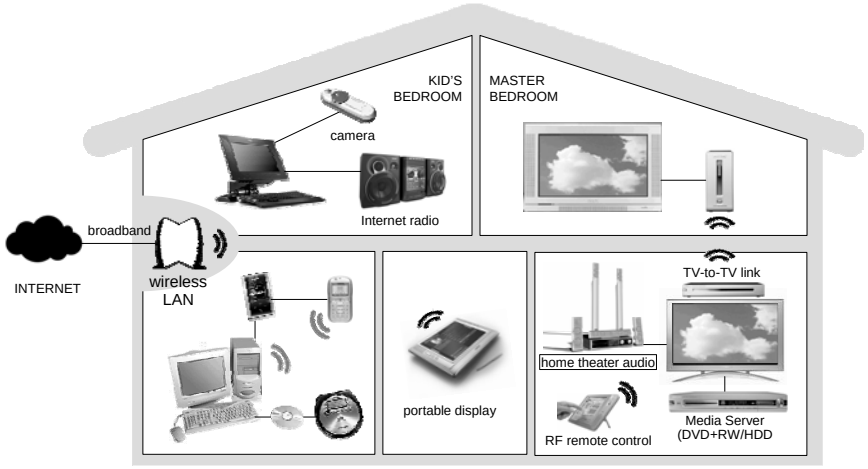



# Feature explosion in consumer products

| | |
|---|---|
| **Imaging** | **Main display** |
| | **Aux display** |
| | **Camera** |
| | **3D display** |
| | **3D rendering** |
| **Audio** | **audio codecs** |
| | **ring tones** |
| | **MP3** |
| **Wireless** | **GSM / GPRS modem** |
| | **UMTS / 3G** |
| | **Bluetooth** |
| | **FM tuner** |
| | **USB** |
| | **GPS** |
| | **RF-ID** |
| | **WLAN** |
| | **DVB tuner** |
| **Wired** | **USB** |
| **Storage** | **Removable Flash** |
| | **Small HDD** |
| | **fixed flash** |
| **SW features** | **Setup & Control** |
| | **Instant messaging** |
| | **PDA** |
| | **Java** |
| | **2-D game** |
| **Video** | **Mpeg2** |
| | **Mpeg4** |
| | **H264** |
| | **DivX** |

## Convergence of digital consumer products




The connected home



## Some of today's cars … ☺

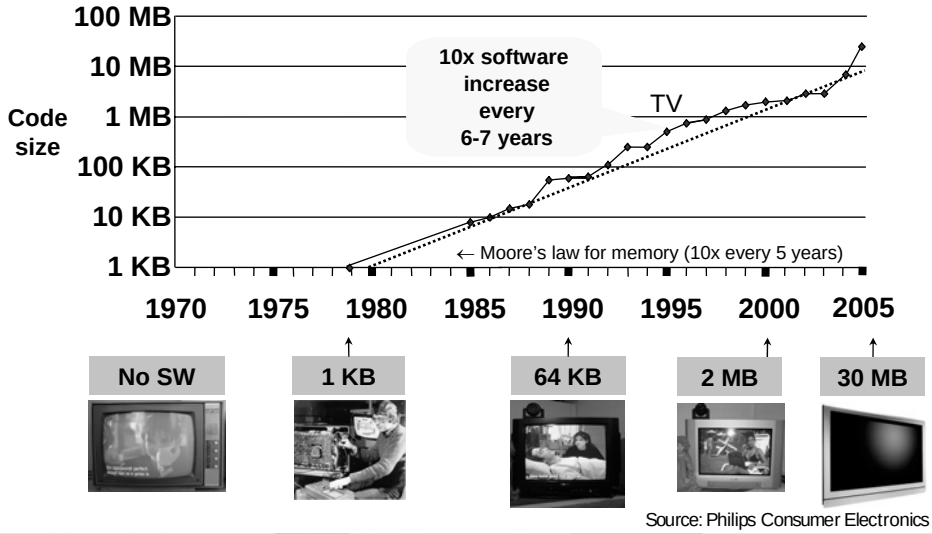

***http://www.aide-eu.org/pdf/aide_nf_050120_engstroem.pdf***

## Convergence of digital consumer products

Consequences:

- Large number of use cases
- Increasingly complex use cases
- Fast changing use cases with new applications appearing
- Format updates at a much higher rate than the device replacement rate
- Uncertain and diversified markets
  - Late / changing product specs, short product life cycles
  - Different customers / tiers have different requirements
- Move from implementations in hardware to software to cope with the variation and changes

## Exponential growth of SW in consumer products




Source: Philips Consumer Electronics

## Consumer electronics

**Characteristics**

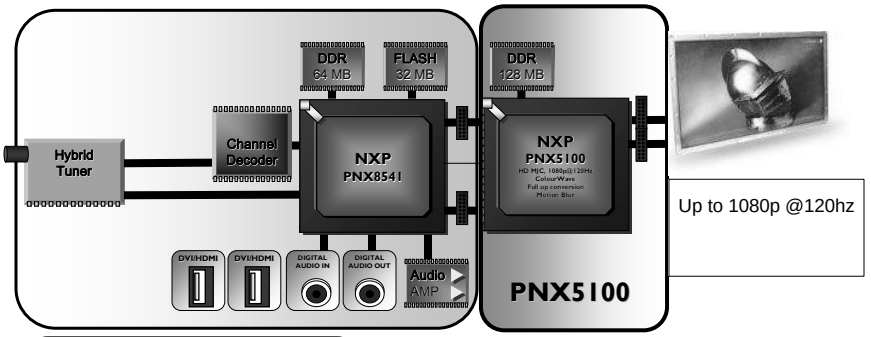
- Cost pressure (< $100 for electronics of large TV)
- Low power consumption (no fan / mobile)
- Large series (> 0.5 million pieces)
- Robustness (no hazards, no reboots)
- Both control and signal processing
- Real-time constraints (no loss of data, guaranteed response)
- Increasing requirements for computation & communication (more functions, higher resolutions)



## Consumer domain demands SoC solutions

- Highly integrated Systems-on-Chip to satisfy constraints on
  - Cost
  - Power consumption
  - Form factor
  - Ease of system integration
- Enabled by large volumes
  - To amortize high NRE of SoC development
  - Need to target sufficiently large market
- Optimized implementations with scarce resources
  - Memory, bandwidth, compute cycles

# Hybrid TV system




Up to 1080p @120hz

Complete one chip TV:

- Cost sensitive midrange market
- Connectivity,OSD
- Hybrid source decode
- De-interlacing
- Audio/Video processing
- Some video featuring

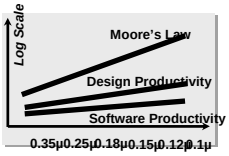
Add on Picture Quality booster

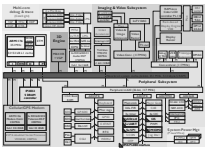
- Halo reduced HD Natural Motion
- LCD Motion Blur Reduction
- Display color and Contrast enhancements

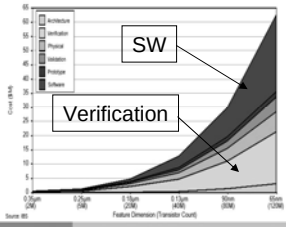


# Key industry trends – More Challenges !

- Cost of system design &integration is getting out of hand
  - Hundreds of man-years for HW/SW system
  - Diversity of products
- Increasing complexity
  - At application, system, HW, and SW level
  - More (sub-)systems on one SoC
  - Single company cannot excel in all domains
- Loosing grip on product quality
  - Exploding costs for verification
  - Degrading reliability and predictability
- More (programmable) cores in single SoC
  - Multi-core programming environments & debug





## Design Solutions

NXP requires a design process that is

- Predictable in time and performance
- Efficient and high quality

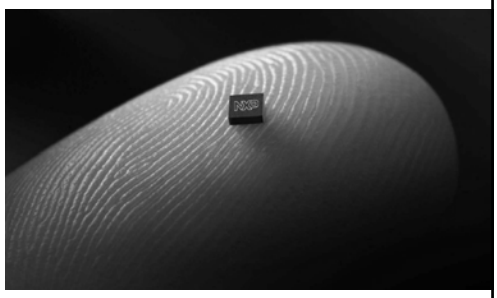
The key elements of such a design process are

- Raising the level of abstraction
  - System Level Design, System Integration and Verification
- High level of re-use
  - IP reuse (HW, SW)
  - Verification reuse
  - Architecture reuse (HW,SW, appl.tasks) → Nexperia platforms
- Integrated design environments
  - Automation of design flow → Builder tools

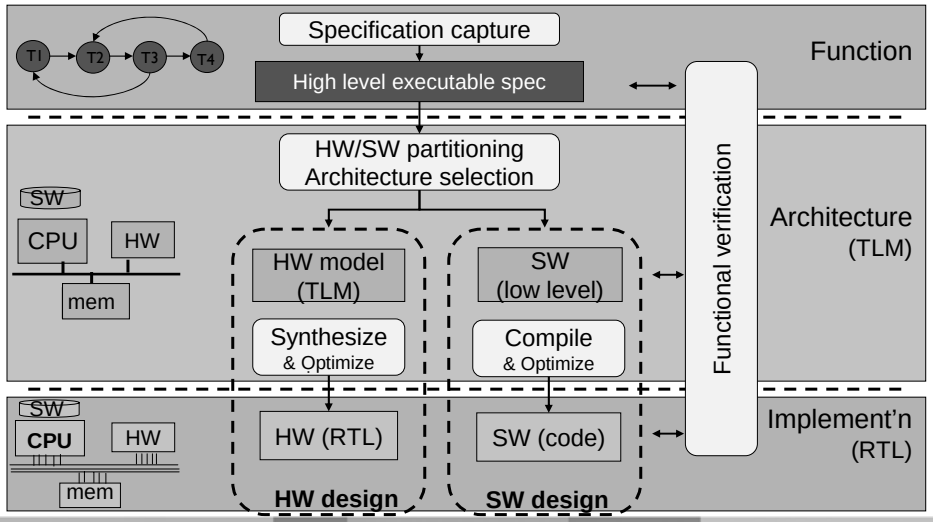


## Outline

- NXP Products / challenges
- **Abstraction Levels**
- Functional Modeling
- Architecture Modeling
- Summary

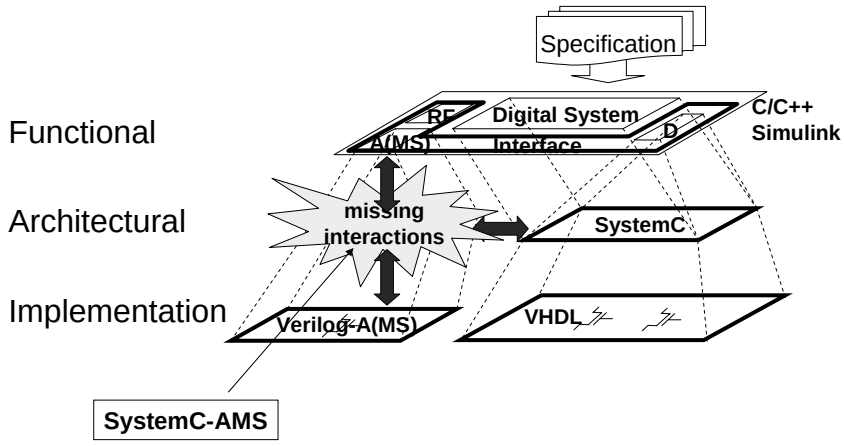


## Design abstraction levels



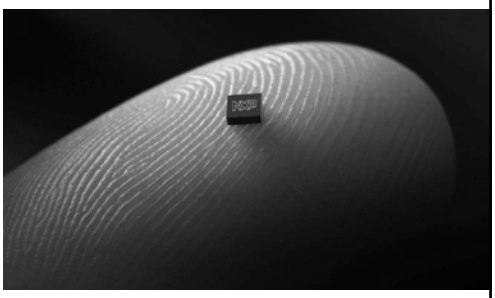



## Design abstraction levels – System-AMS



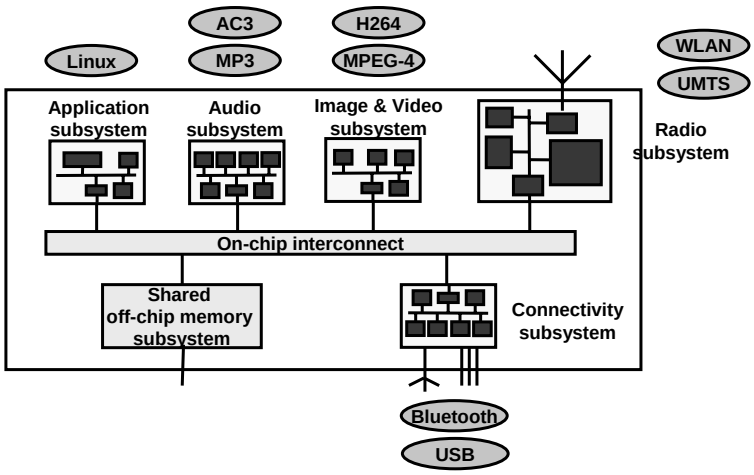
# Outline

- NXP Products / challenges
- Abstraction Levels
- **Functional Modeling**
- Architecture Modeling
- Summary





# NXP SoC Example (simplified view)

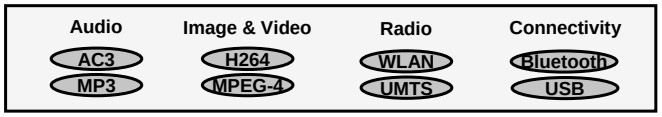



**A SoC is composed of coarse-grain SubSystems**

## SoC Level Functional Model

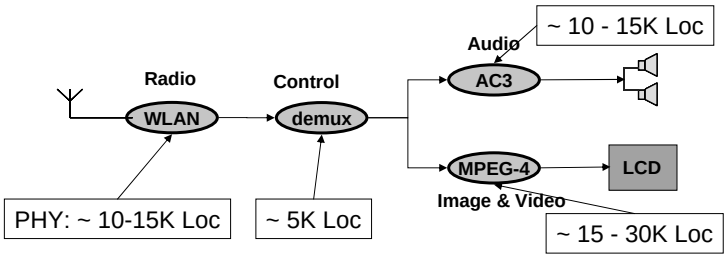
Why do we need a functional model?

- Functional verification of algorithm
- Single model for HW and SW parts.
- Explore algorithmic tradeoffs
- Estimate resource requirements
  - Computational load, Communication load






## SoC Level Functional Model (toplevel)

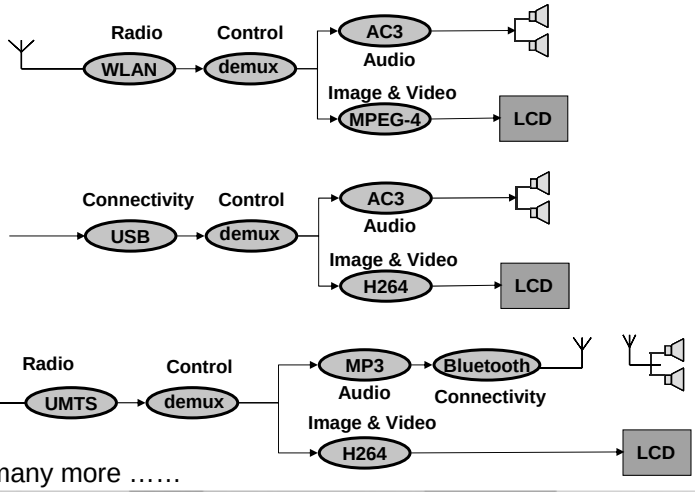



Difficult to create an executable SoC level functional model

- Each sub-function contains hundreds of sub-blocks
  - Functional decomposition, Interfaces
- Characteristic per functional sub-system differ
  - Different semantics needed
    - Dataflow, Process networks, Discrete Event, State Machines
  - Different algorithmic tradeoffs

# SoC Level Functional Model – Use Cases

Many (complex) application use cases to handle

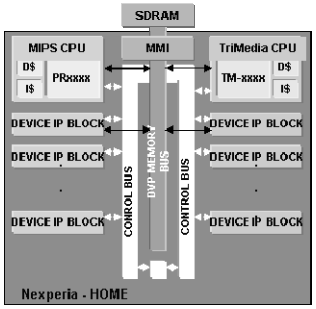



And many more ……



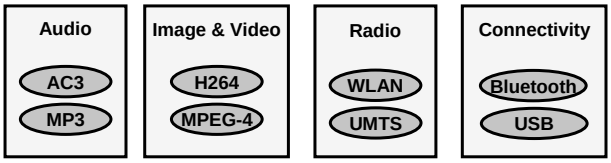
# SoC functional model - Hurdles

- Different Models of Computation (semantics)
- Single functional model too complex to handle by single designer
- Limited tool support
- Many application use case to cover
  - "All in one" functional model not preferred
- IP Re-use, platform based design
- Many design teams involved
  - Multi site, Multi culture




Solution : Divide & Conquer approach

# Functional Modeling at IP / Sub-System Level



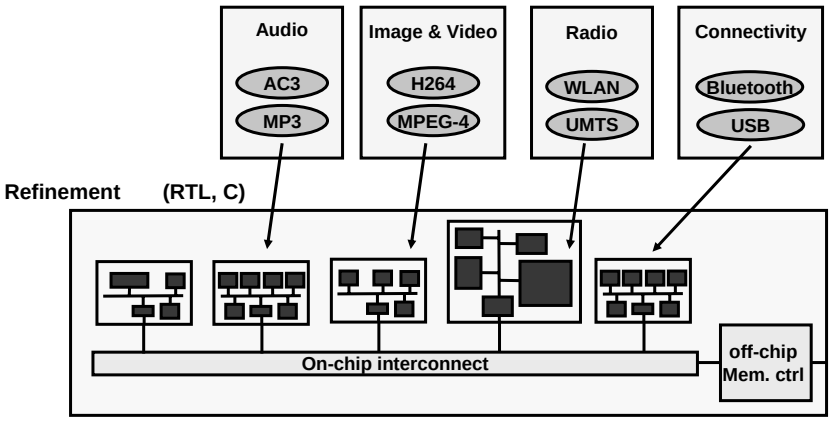

Benefits of multiple functional models:

- Complexity can be handled more easily
- Optimized tools available for implementations
- Specific implementation technology
  - Domain specific, single semantics
- Specialized domain know-how (e.g. competence centre)
- Different life-cycles → re-use

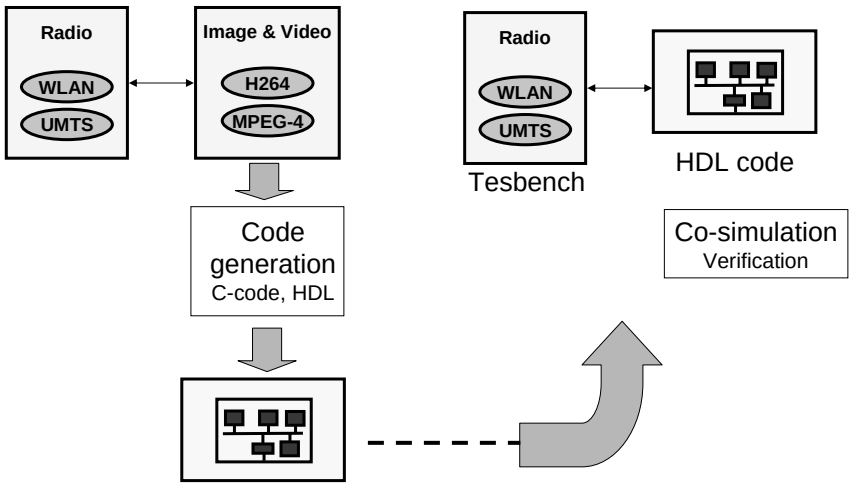


# Functional Model Refinement

- Functional refinement directly to implementation level



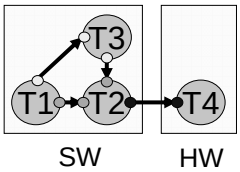
## Functional Refinement
Code-generation & Co-simulation





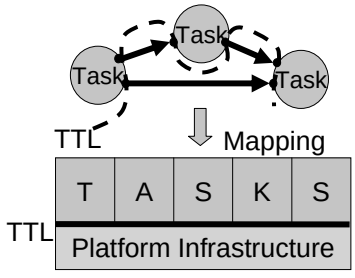

## Functional Modeling – Multimedia Domain

- An interface centric design approach
  - Communicating tasks based on KPN
  - Optimal SW and HW interface types



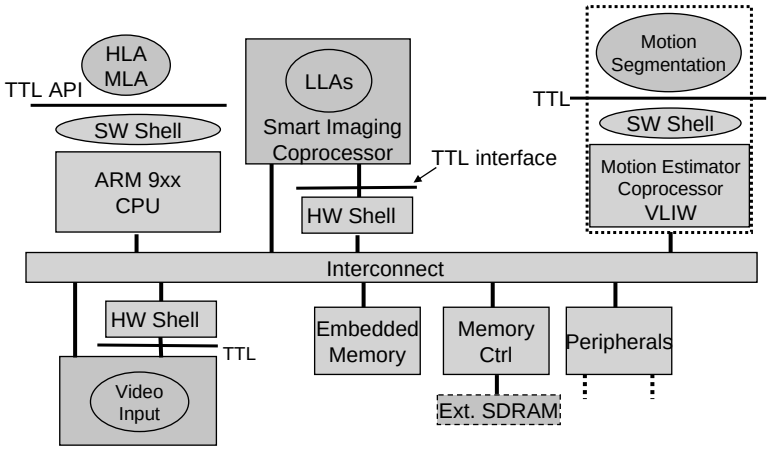

Aim: Close gap between specification and implementation

- TTL Task Transaction Level interface:
  - Parallel application models
    - Executable specifications
  - Platform interface
    - Integration of HW and SW tasks
- Mapping technology
  - Structured design & programming




[CODES04]

## Example TTL based designs
Smart Imaging Core



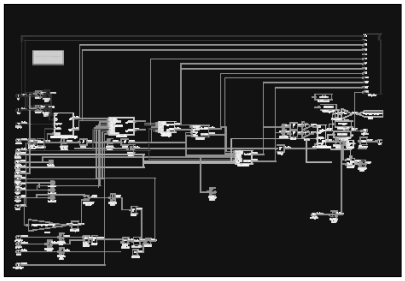

[CODES05] , [DAES06]

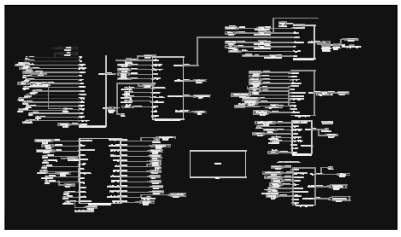


## Functional modeling - Simulink
Bluetooth - EDR

- Started with Simulink built-in models.
- Developed a complete TX chain, including analog/RF blocks.
  - Simulated the transfer function of the modulator and verified it against the Bluetooth standard specification
- Converted the design into fixed-point format .
  - Verified the functionality of the fixed-point design against the Bluetooth standard.
- Generated (manually) a bit-compatible RTL description of the Simulink fixed-point model.
- Evaluated the modulator on an FPGA platform.

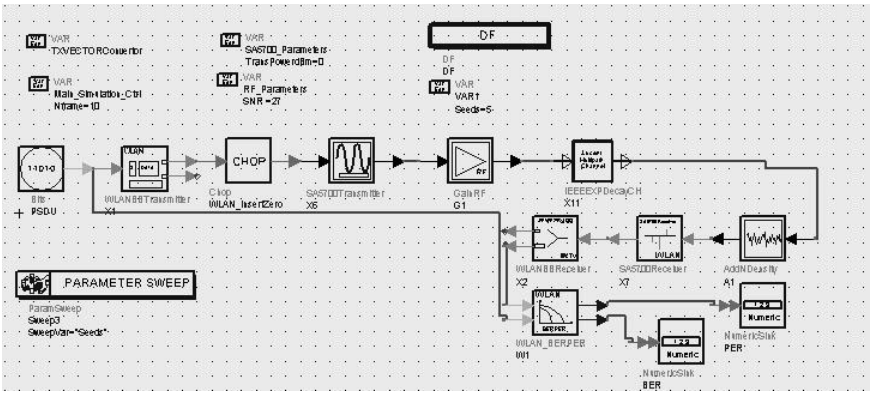
Next step: HDL generation directly from Simulink





## Functional modeling - ADS/Ptolemy

Wireless LAN radio system




- Functional model used as testbench for circuit design



## What's next after functional modeling?

Functional to implementation problems

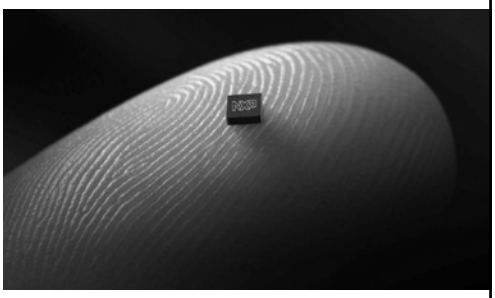
- SoC integration happens at RTL / C level
  - Complex and error prone
  - Iterations are not possible (first implementation/configuration that work instead of best/optimized one)
- Main usage of the Implementation level is for pre-silicon verification
  - But complete system is hard to simulate/verified all together
- SoC integration at the implementation level introduces a big gap
  - Complex and error prone
  - Iterations are not possible (first implementation that work instead of best one)

An intermediate step before implementation is required

➔ Architecture level
  - Main usage is SoC integration, analysis and debug

# Outline

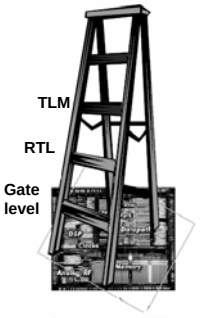
- NXP Products / challenges
- Abstraction Levels
- Functional Modeling
- **Architecture Modeling**
- Research challenges
- Conclusions

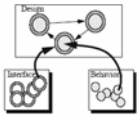


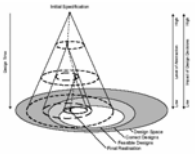


# Architectural Level – Key concepts

- Abstraction
  - Increasing block granularity
    - Gates (80's), HDL (90's), IP blocks (2000), SubSystems (2010)
  - From RTL to TLM (Transaction Level Modeling)
    - but TLM covers only HW architecture, all system aspects need to be modeled together (i.e. architecture, application, constraints, etc)
- Separation of concerns
  - Computation / Communication / Cost
  - Behavior / interfaces
- Refinement
  - Synthesis and transformation techniques
- Standards
  - OSCI (SystemC and TLM standards)
  - SPIRIT (IP-XACT)

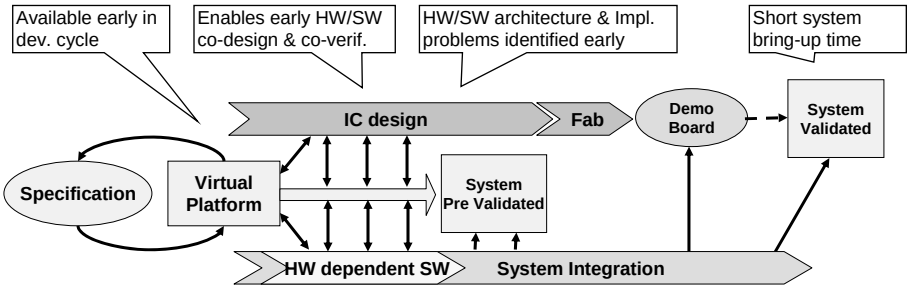


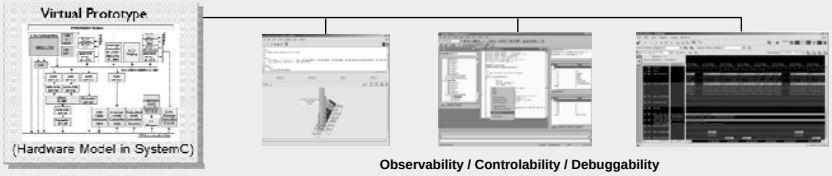


SYSTEMC™ IP-XACT

## Architectural model => Virtual prototype



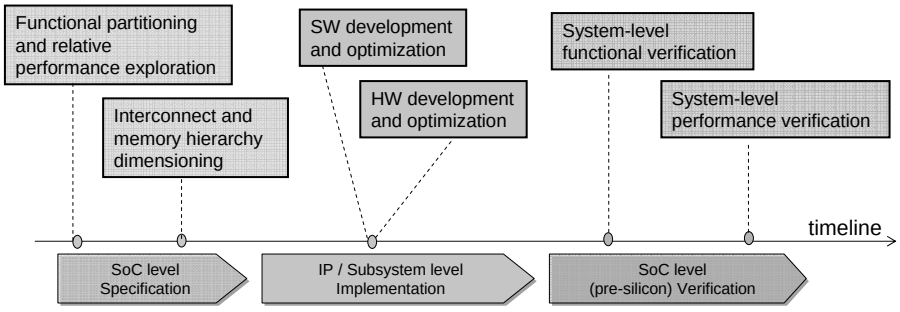

**Technology Implementation:**




**Observability / Controlability / Debuggability**



## But more than one architectural model is required




Use-case driven approach

- Different activities per design phase
  - Specification: exploration and dimensioning (SoC level)
  - Implementation: SW and HW development and optimization (IP and SubSystem level)
  - Verification: functional and non-functional verification of both HW and SW together
- Activities have different requirements

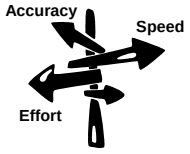
# Different use-cases, different requirements

| Use-case | | Description | RTL speed up | Behavior accuracy | Timing accuracy |
|---|---|---|---|---|---|
| **Functional partitioning and relative performance estimation** | | ▸ Enable extensive architectural exploration (what-if scenarios)<br>▸ Focus on: task mapping, task scheduling, memory allocation, resource allocation, etc | $x10^4$ | Mixed abstract performance and functional models | Relative (fidelity) |
| **Interconnect and memory hierarchy dimensioning** | | ▸ HW architecture dimensioning<br>▸ Focus on: bandwidth, latency, buffering, arbitration policies, etc | $x10^2$ | | Absolute ±10% |
| **SW development and optimization** | **Application** | ▸ End-user SW applications (platform services based)<br>▸ Multicore debugging | $x10^4$ | Fully functional models | Absolute ±20% |
| | **Middleware** | ▸ Complex algorithms and codecs (H264, MP3, UMTS, AES etc)<br>▸ SW platform services | $x10^3$ | | Absolute ±10% |
| | **HW dependent** | ▸ Low-level drivers, HW abstraction layers (HAL)<br>▸ Must be very optimized | $x10^2$ | | Absolute ±1% |
| **HW development and optimization** | | ▸ VP as a system-level test-bench for the HW IP<br>▸ High Level Synthesis / RTL co-simulation (transactors) | $x10^3$ | | Absolute ±10% |
| **System-level functional verification** | | ▸ HW/SW integration and verification (complete SW stack)<br>▸ Extensive verification of scenarios | $x10^3$ | Fully functional models | Absolute ±10% |
| **System-level performance verification** | | ▸ Very accurate performance and power estimations<br>▸ Representative (worst-case) scenarios | $x10^2$ | | Absolute ±1% |



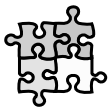
# Different requirements, different models?

- In an ideal world one model fits all
  - Ultra fast models + 100% cycle accurate + push-button availability
- Unfortunately today we still need the right model for the right use-case… but one model per use-case is an overkill
  - Huge effort to create and maintain every model
  - How to assure consistency between models?
- Model reuse and refinement is a must… but modeling is a multidimensional problem
  - High speed models contains very little time information
  - Very accurate models are typically slow and are difficult to create
  - No clear refinement paths from one model to another
- And different types of models have different particularities
  - Processing units (CPU, DSP, etc)
  - Interconnects (busses, bridges, routers, etc)
  - Memory subsystem (L1-L2 cache, memory controllers, memories, etc)
  - Peripherals (IO blocks and slave HW accelerators)




Getting to the right model is the critical task

# Modeling concepts



- Minimizing the modeling effort
  - Library of predefined building blocks to compose IP models
  - Well defined methodologies and guidelines
  - Standard set of IP interfaces to assemble IP models (avoid adaptors)

- Modeling for speed
  - Minimize the number of simulation events / context switches
  - Coarse grain computation and communication
  - Binary translation techniques for processors / host-code emulation techniques

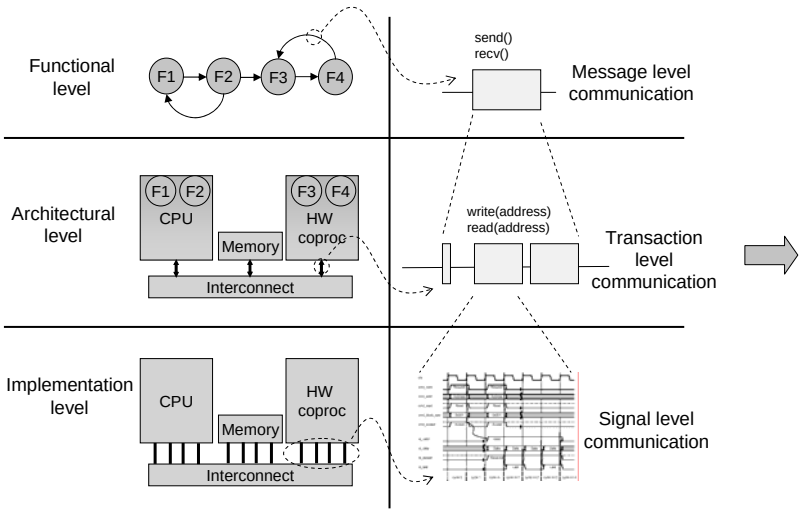
- Modeling timing
  - Accurate timed models are very hard to create
  - Approximated timed models are easier (but when it is good enough? )
  - Define accuracy windows, where the window size depends on the use-case
- Refinement
  - Interfaces/Communication refinement ➡ Being tackle by OSCI TLM 2.0 standard proposal
  - Behavior/Computation refinement
  - Cost (timing) refinement

  } ➡ Still ad-hoc, proprietary solutions

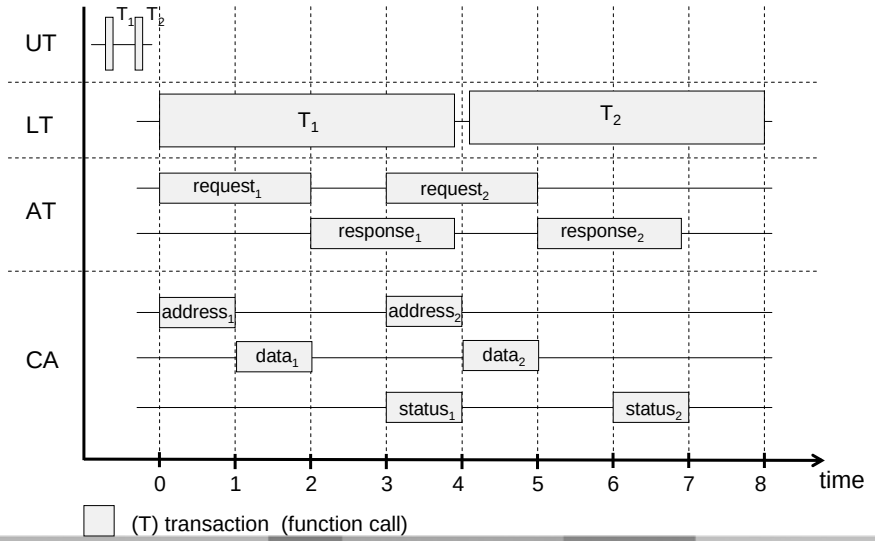
[TLM]



# Abstraction Levels with SystemC



# OSCI TLM 2.0 modeling styles






# How many models do we need?

| Use-case | | Processor units | Interconnects | Memory subsystems | Peripherals |
|---|---|---|---|---|---|
| **Functional partitioning and relative performance estimation** | | ▸ High-level tasks<br>▸ Mapping technology | ▸ Functional models (AT), protocol independent | | ▸ High-level tasks<br>▸ Mapping technology |
| **Interconnect and memory hierarchy dimensioning** | | ▸ Traffic generators | ▸ Functional models (CA), protocol specific | | ▸ Data sinks |
| **SW development and optimization** | **Application** | ▸ Fully functional models at the UT, LT level (Proc. IA ISS) | | | |
| | **Middleware** | ▸ Fully functional models at the LT, AT level (Proc. IA ISS) | | | |
| | **HW dependent** | ▸ Fully functional models at the CA level (Proc. CA ISS) | | | |
| **System-level functional verification** | | ▸ Fully functional models at the LT level (Proc. IA ISS) | | | |
| **System-level performance verification** | | ▸ Fully functional models at the CA level (Proc. CA ISS) | | | |

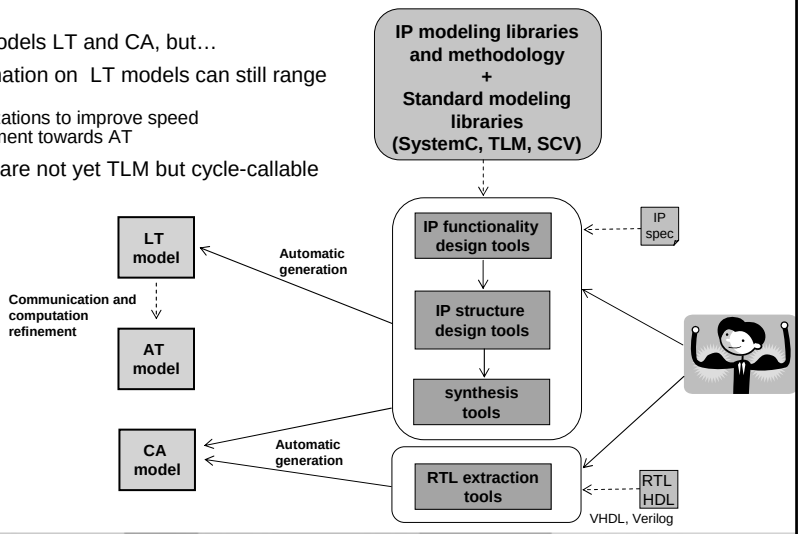
Buy, outsource

In-house creation

# Creating efficiently functional IP models

2 functional models LT and CA, but…

- Time information on LT models can still range quite a bit
  - Optimizations to improve speed
  - Refinement towards AT
- CA models are not yet TLM but cycle-callable signal-level

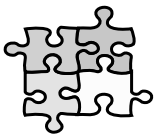




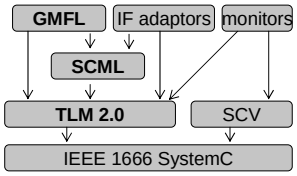

# IP modeling libraries and methodology

Objectives:

- Reduce the modeling effort and learning curve for new model developers
  - Quickly create the most common parts of the IP model using building blocks

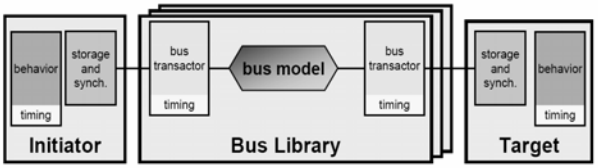


- Enable model reuse and refinement
  - Add details gradually (both communication and computation)
  - Separation of concerns applied to modeling
- Improve configurability of models
  - Change model internals during construction/elaboration time
- Align with industry standards
  - *OSCI TLM 2.0 & IP-XACT 1.4*




# CoWare SCML


- Methodology for model re-use
  - Use-case determine for bus model
  - Re-use peripheral model for multiple use-cases
- Separation of behavior, communication and timing
- Focus on communication refinement
  - generic interfaces + specific adaptors
- Simple modeling pattern supported by a library of predefined blocks
  - SystemC Modeling Library (SCML) http://www.coware.com/solutions/tlm.php

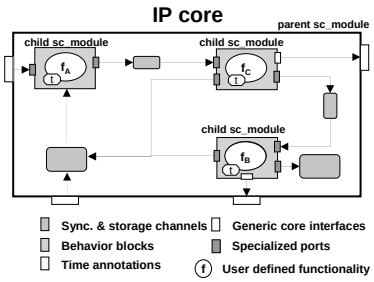





# NXP GMFL

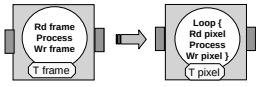
Generic Modeling Features Library

- Library of predefined blocks built on top of SCML
- Extend and complement SCML capabilities
  - More methodology and guidelines
- Focus on behavior modeling and refinement
  - Explicit synchronization, data and control flow




Key concepts are:

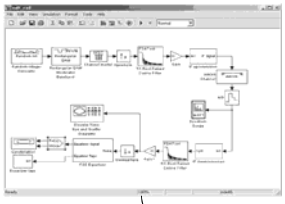
- Hierarchical modeling
  - Structured, reusable code
  - Closer to HW designers
- Dynamic layout
  - Add/remove functionality during elaboration
  - Block refinement



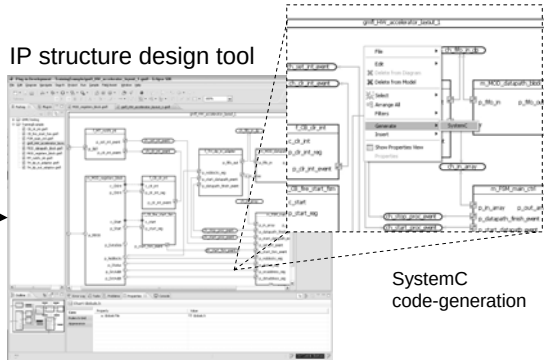

- Design and code-generation tool

# NXP GMFL design and code-generation tool

Functional design tool (3rd party)



IP structure design tool

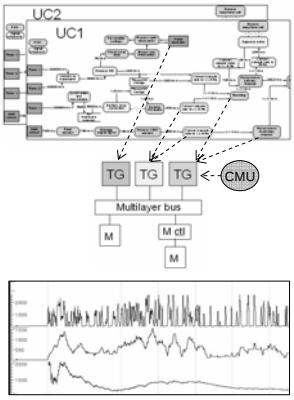


import

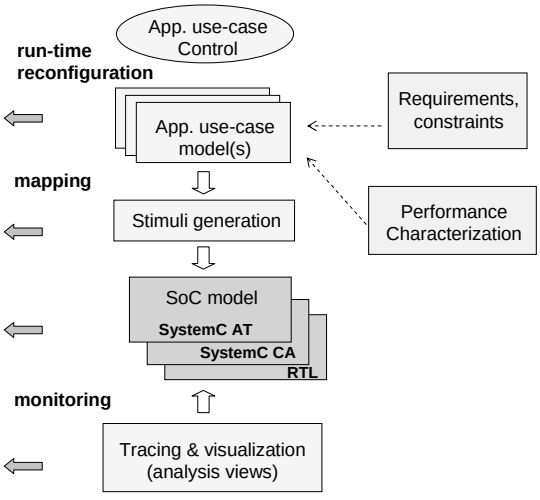
SystemC
code-generation



# Abstract (dynamic) performance simulations

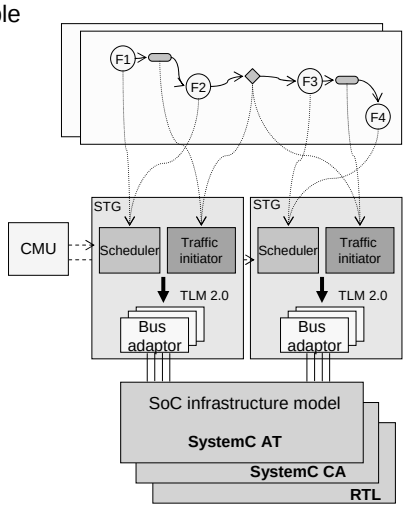




# Application use-case modeling and mapping

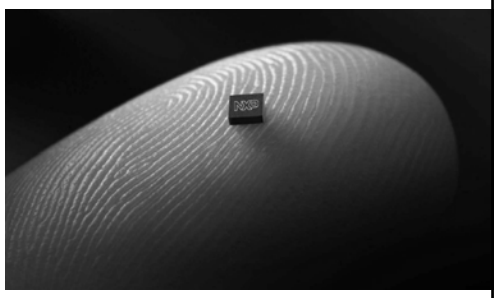
- Executable application use-case model (simple semantics)
  - Synchronization & data communication
  - Dynamic data dependencies
  - Constraints (throughput, latency, etc)
- Smart Traffic Generators (STG)
  - Direct mapping from app. use-case models
  - High level semantics adopted to bus protocol traffic
  - Possibility to simulate
    - **function/data dependencies**
    - **Synchronization between STG**
    - **Scheduling / interrupts / preemption**
  - Generic interface + specific adaptors
- Configuration Manager Unit (CMU)
  - Multiple use-cases can be mapped on a STG
  - CMU on charge of **use-case switching**






# Outline

- NXP Products / challenges
- Abstraction Levels
- Functional Modeling
- Architecture Modeling
- **Summary**



## Summary

- SoC design challenges require moving up in the abstraction level and exploit re-use at all levels
- Abstraction levels
  - Multiple domains (digital, analogue, RF)
  - Multiple languages (UML, C++, Simulink, SystemC, VHDL, Verilog-AMS, etc)
- Functional modeling → divide & conquer
  - Domain specific functions, different semantics, different algorithmic trade-offs
  - Multiple functional use-cases
  - Most tools provide a path to implementation
- An integration level above implementation is required → Architectural Level
  - Architecture model => Virtual prototype
  - Different architectural use-cases with different requirements => the right model for the right use-case
  - Getting to the models is still the critical task => how to tackle this
    - IP Modeling methodologies and libraries
    - Automatic generation from different tools



## References


- [DAC2000] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J-Y. Brunel, W.M. Kruijtzer, P. Lieverse, K.A. Vissers; *"YAPI: Application Modeling for Signal Processing Systems "* in Proceedings of the 37th Design Automation Conference, Los Angeles, 2000
- [CODES04] Pieter van der Wolf, Erwin de Kock, Tomas Henriksson, Wido Kruijtzer, Gerben Essink, *"Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach"*, Int. Conf. on Hardware/Software co-design and System synthesis, September 2004
- [CODES05] Wido Kruijtzer, Winfried Gehrke, Victor Reyes, Ghiath Alkadi, Thomas Hinz, Jörn Jachalsky, Bruno Steux; *"The Design of a Smart Imaging Core for Automotive and Consumer Applications: A Case Study"* in Int. Conf. on Hardware/Software co-design and System synthesis, September 2005
- [DAES06] Wido Kruijtzer, Victor Reyes, Winfried Gehrke; *"Design, Synthesis and Verification of a Smart Imaging Core using SystemC"* in Design Automation for Embedded Systems, An International Journal from Springer, Volume 10, Numbers 2-3, September 2006, pp. 127-155(29). Special Issue on SystemC-based System Modeling, Verification and Synthesis. DAC07: Slider
- [DAC2007] Walter Tibboel, Victor Reyes, Martin Klompstra, Dennis Alders ; "System-Level Design Flow Based on a Functional Reference for HW and SW" in Proceedings of the 44th Design Automation Conference, San Diego, 2007
- [SCML] http://www.coware.com/solutions/scml_kit.php
- [TLM] http://www.systemc.org/downloads/standards/
- [ESL book] Brian Bailey, Grant Martin, Andrew Piziali, *"ESL Design and Verification: A Prescription for Electronic System Level Methodology"* The Morgan Kaufmann Series in Systems on Silicon