Introduction to SoC, Multimedia Systems, and ESL

Shao-Yi Chien

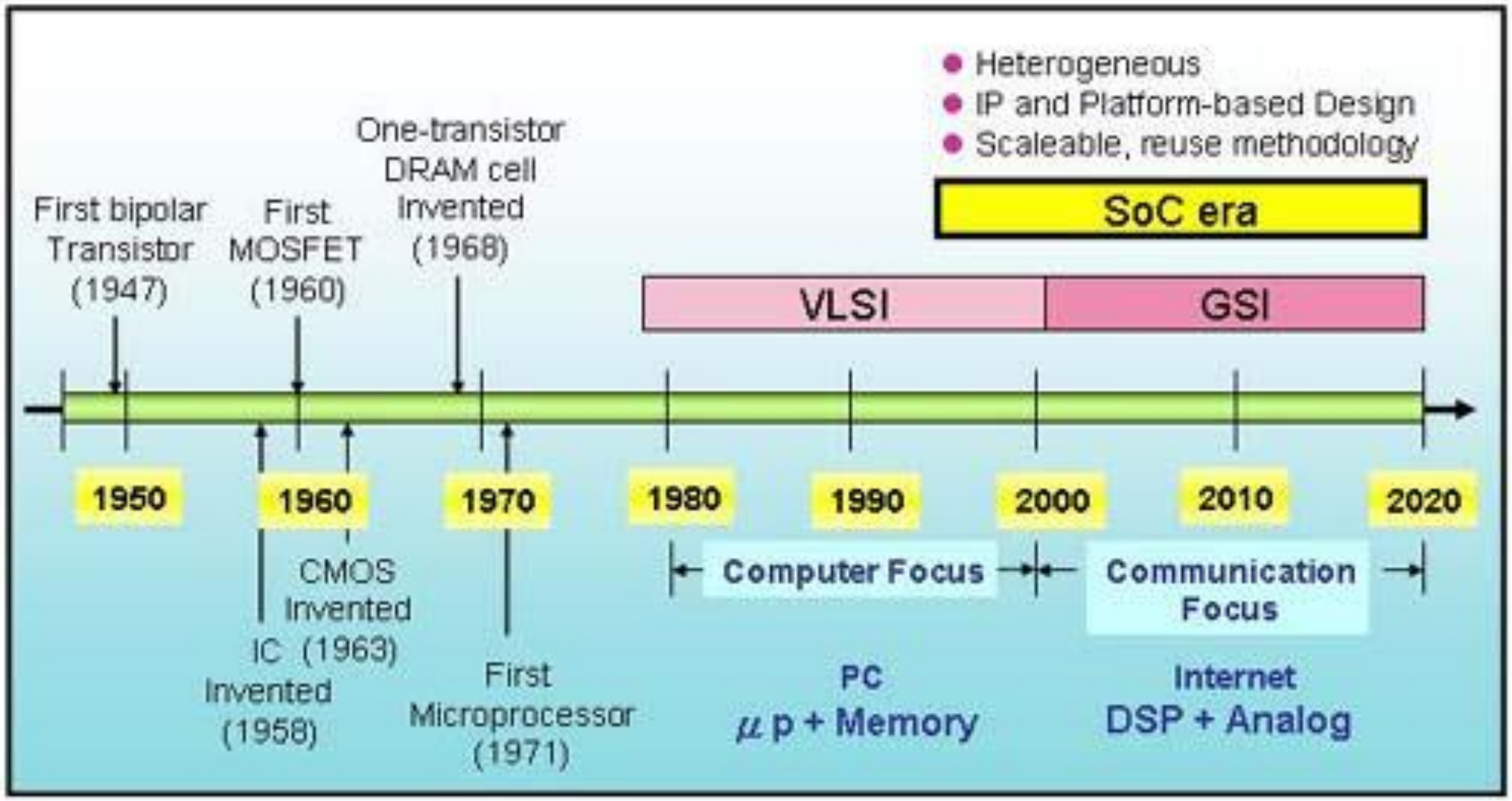
Outline

- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

Outline

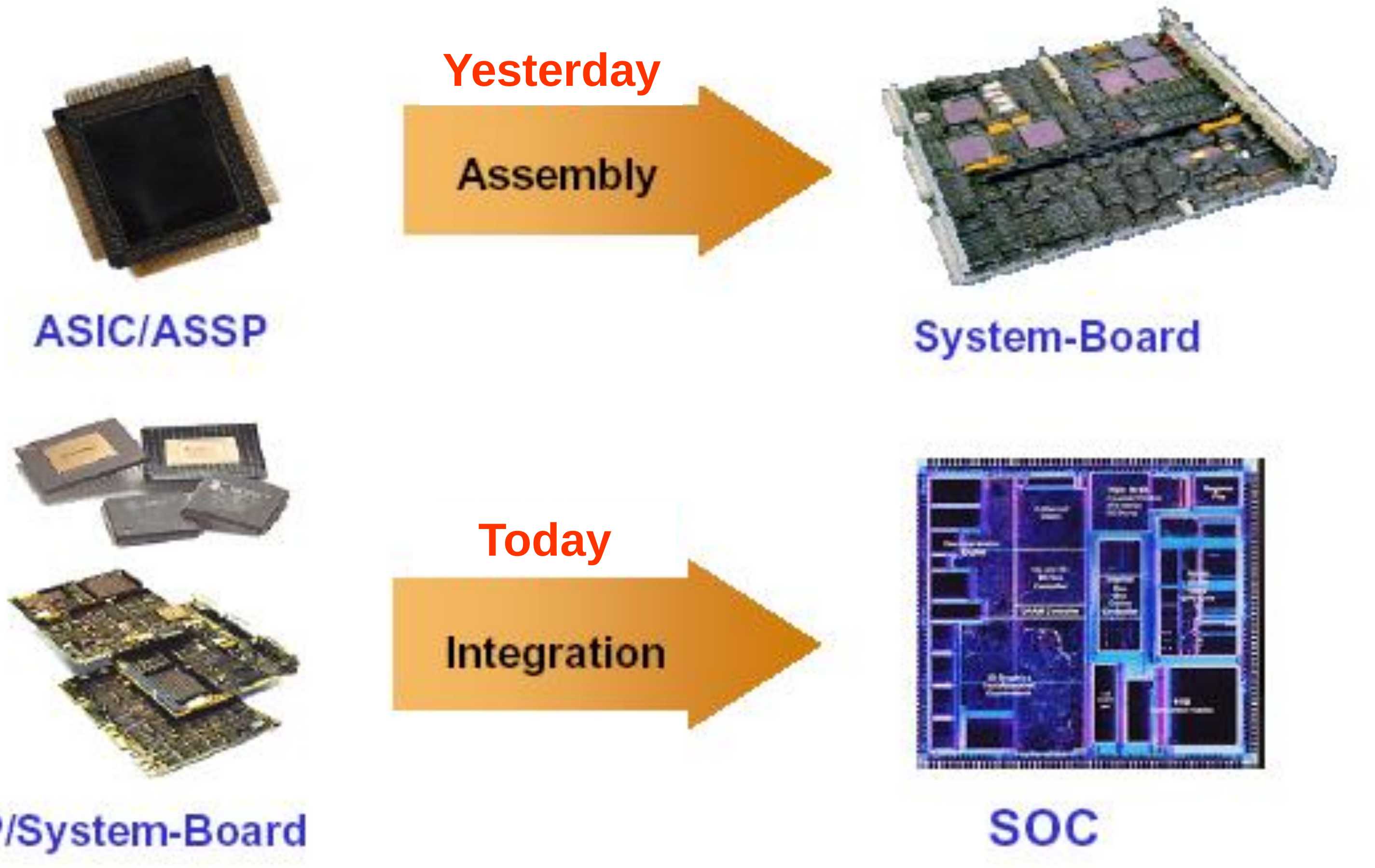
- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

Silicon Evolution

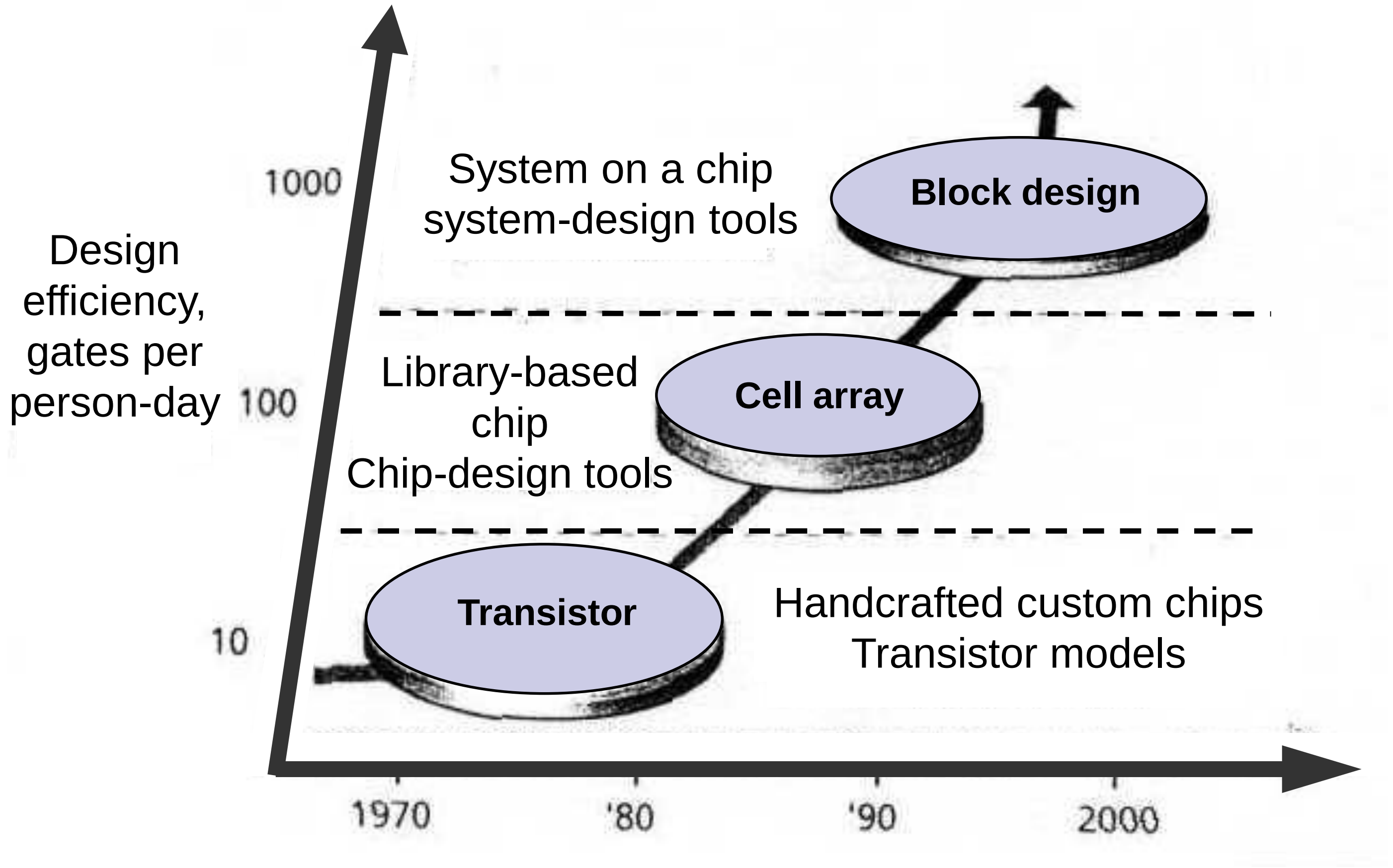


Why System-on-a-Chip?

Design Paradigm Shift



Changes in the Nature of IC Design



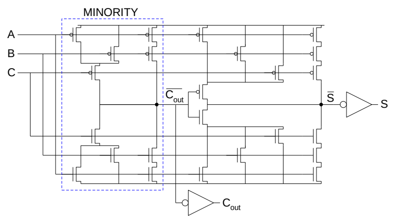
Design efficiency, gates per person-day
1000
100
10
System on a chip system-design tools
Block design
Library-based chip Chip-design tools
Cell array
Transistor
Handcrafted custom chips Transistor models
1970
'80
'90
2000

(IEEE Spectrum Nov,1996)

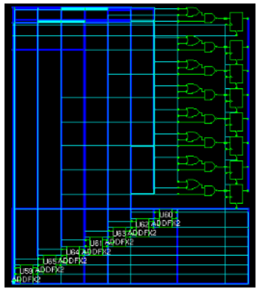
From ASIC to SoC

Yesterday

- HW only
- Perfect interconnection

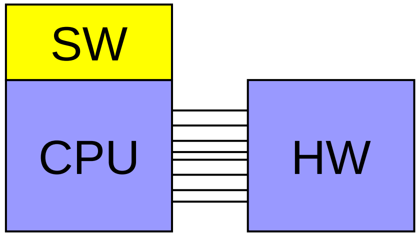


```
module counter(
clk,                    //Clock signal.
reset_n,        //Async. reset signal, active low.
enable,                 //Sync. enale signal. The counter won't act without enable==1'b1.
data                    //output counter data.
);

input clk;              //Clock signal.
input reset_n;                          //Async. reset signal, active low.
input enable;           //Sync. enale signal. The counter won't act without enable==1'b1.
output [1:0] data;      //output counter data.

reg [1:0] data, data_w;

.....

always@(posedge clk or negedge reset_n)
begin
  if(!reset_n)
  begin
    data <= #1 2'd0;
  end
  else
  begin
    data <= #1 data_w;
  end
end

endmodule
```

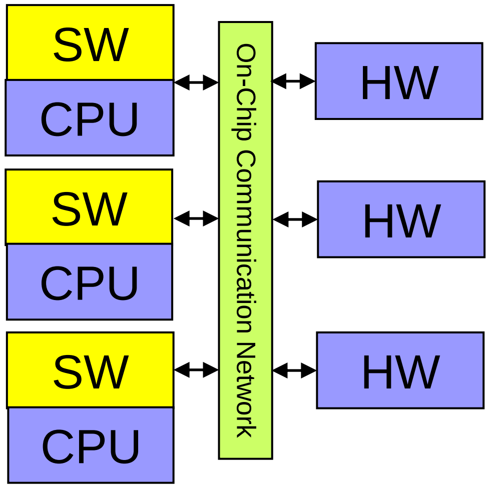


Today

- Heterogeneous
- CPU + dedicated HW



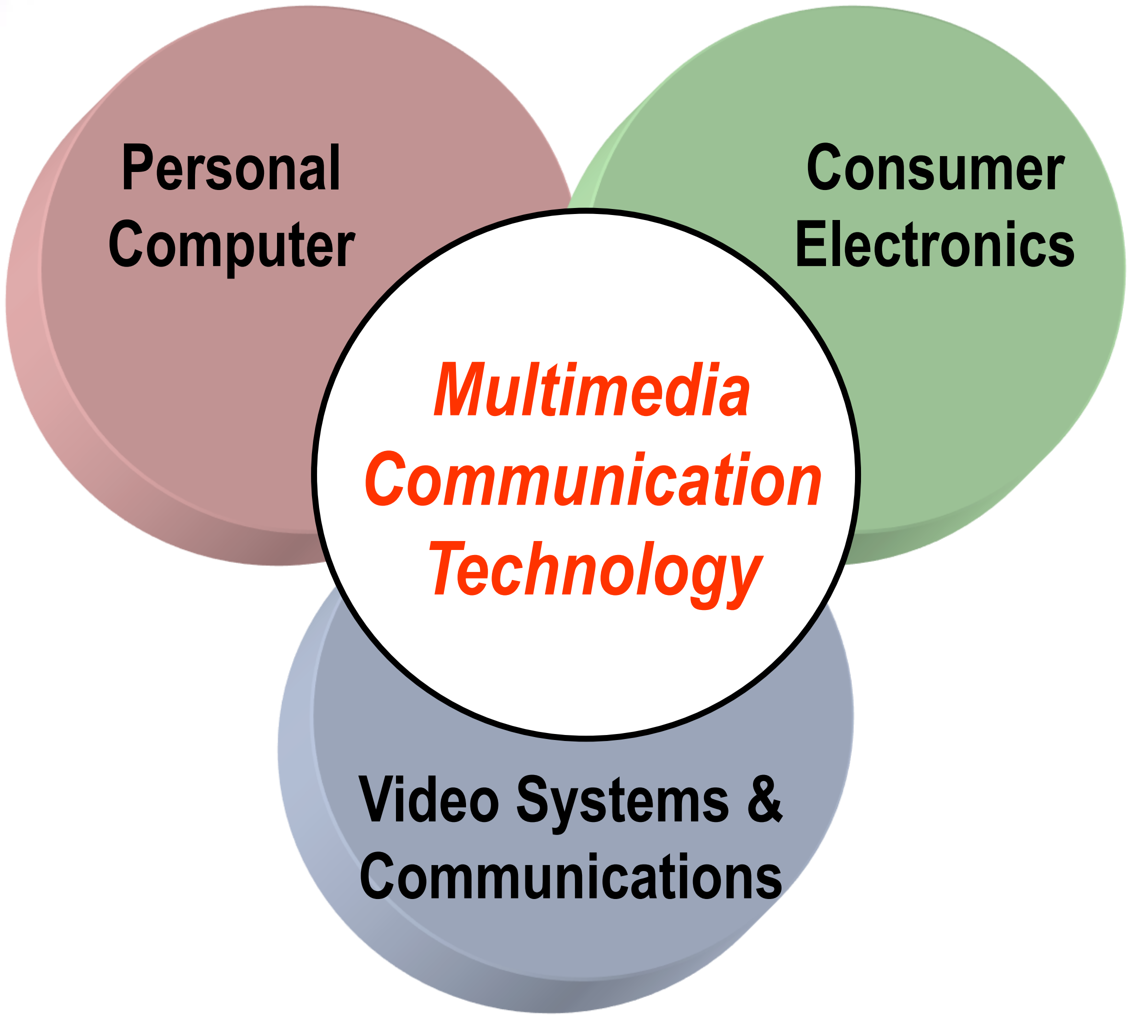
- Multiple SW stacks
- Non perfect interconnect



Outline

- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

Digital Convergence


Personal Computer
Consumer Electronics
Multimedia Communication Technology
Video Systems & Communications

Multimedia Technology for Human Life

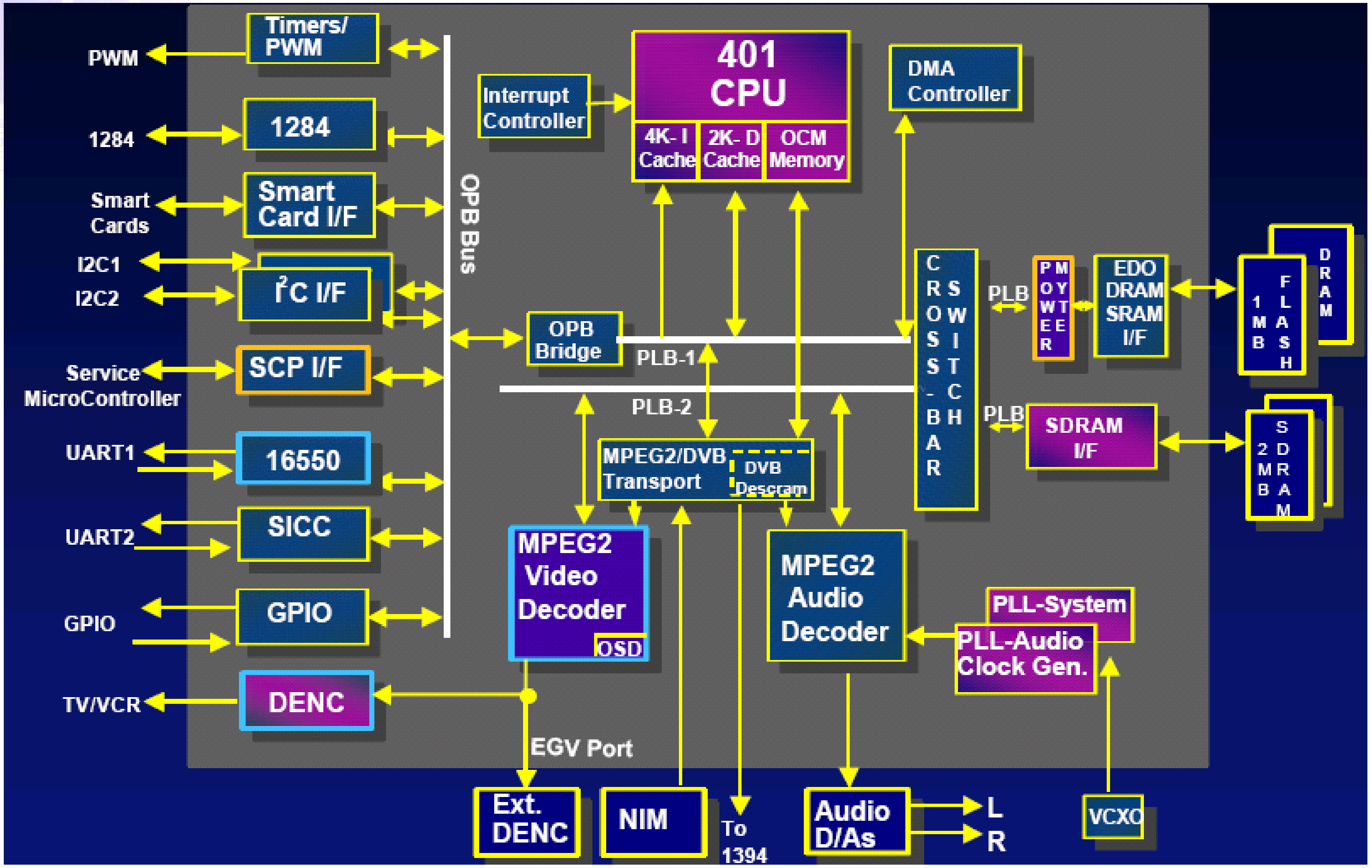
- From office to home and the outdoors
- From large devices to portable devices
- From specific people to everybody

Any Time
Any Where
At Will

Relationship between SoC and Multimedia Systems

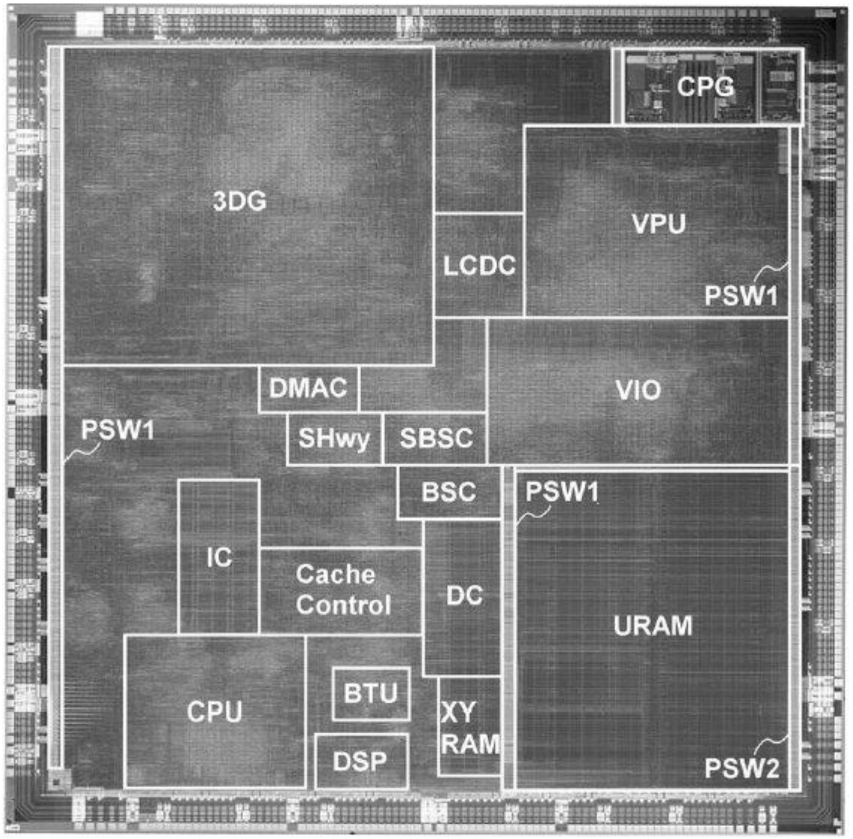
- Multimedia systems integrate many subsystems
  - User interface
  - Image/video/audio capturing
  - Image/video/audio displaying
  - Image/video/audio processing and coding
  - Communication and storage
- High volume of the consumer electronics
- Both the factors make multimedia system a highly possible application for SoC
  - TV/STB, mobile phones, wearable devices, AR/VR, automotive electronics, multimedia players, multimedia portable players, game consoles, …

SoC Example: Set Top Box Controller



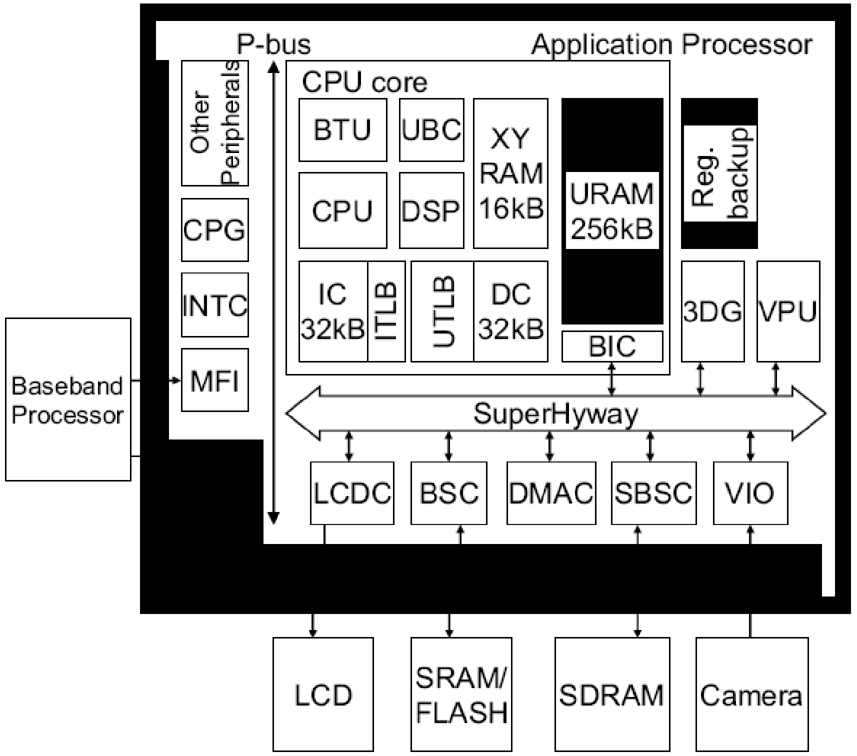
SoC Example: Multimedia Mobile Phones (1)

- Renesas application processor for 3G cellular phones



T. Kamei et al., "A resume-standby application processor for 3G cellular phones," *ISSCC Dig. Tech. Papers*, pp. 336—337, Feb., 2004.

SoC Example: Multimedia Mobile Phones (2)

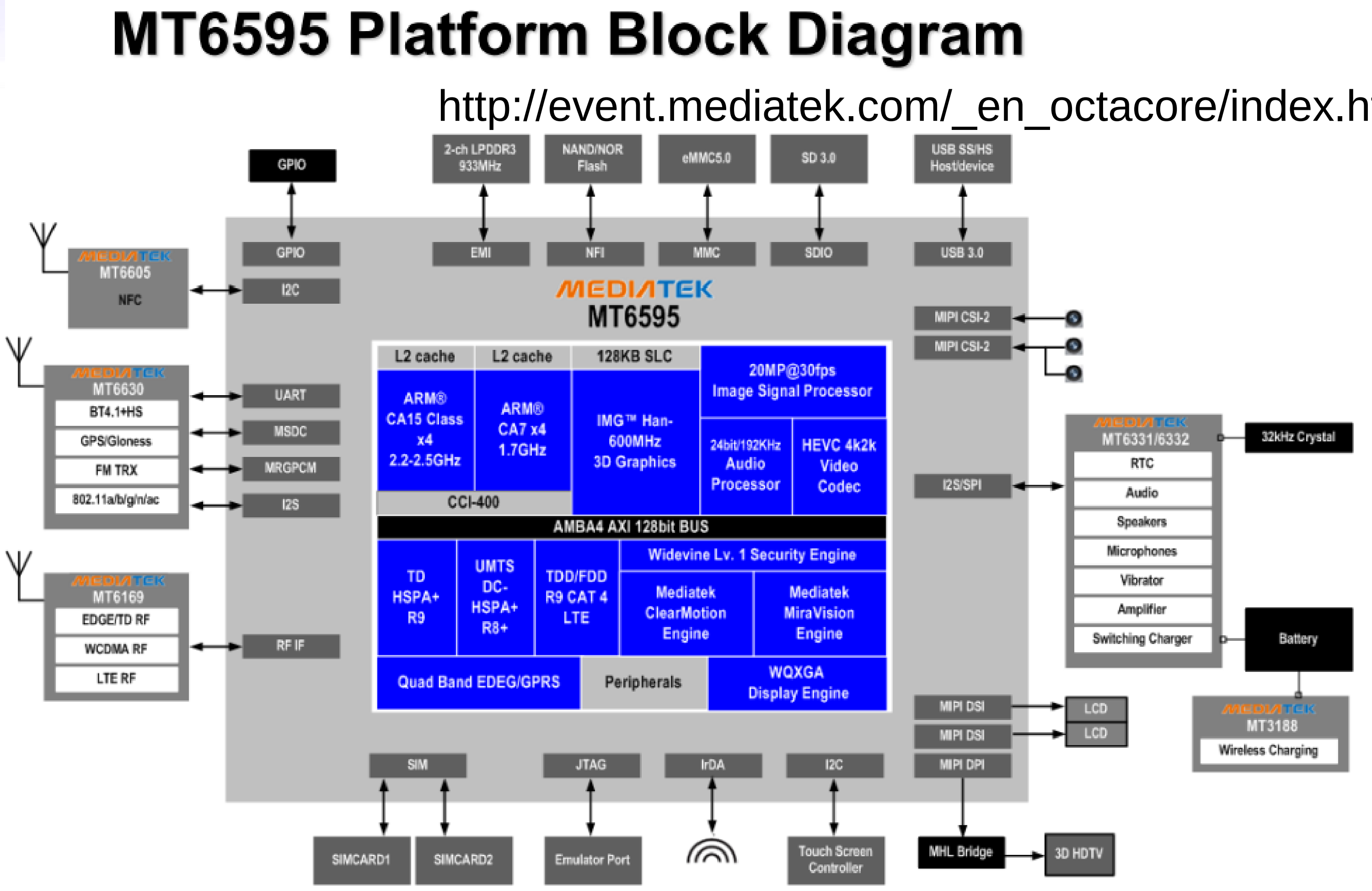


BTU:	Bytecode Translation Unit
BIC:	Bus Interface Controller
BSC:	SRAM/FLASH Controller
SBSC:	SDR/DDR-SDRAM Controller
LCDC:	LCD Controller
VIO:	Camera I/F with Image-Processing Engine
VPU:	MPEG-4 Accelerator
3DG:	3D Graphics Accelerator
CPG:	Clock Pulse Generator
INTC:	Interrupt Controller
MFI:	Multi-Functional Interface
RWT:	Real-time Watchdog Timer
PSC:	Power Supply Controller

- ☐ Area1: inactive in R/U-standby mode
- ■ Area2: active in R-standby mode
- ■ Area3: active in R/U-standby mode
- ■ Area4: I/O cells(1.8V or 3.3V)

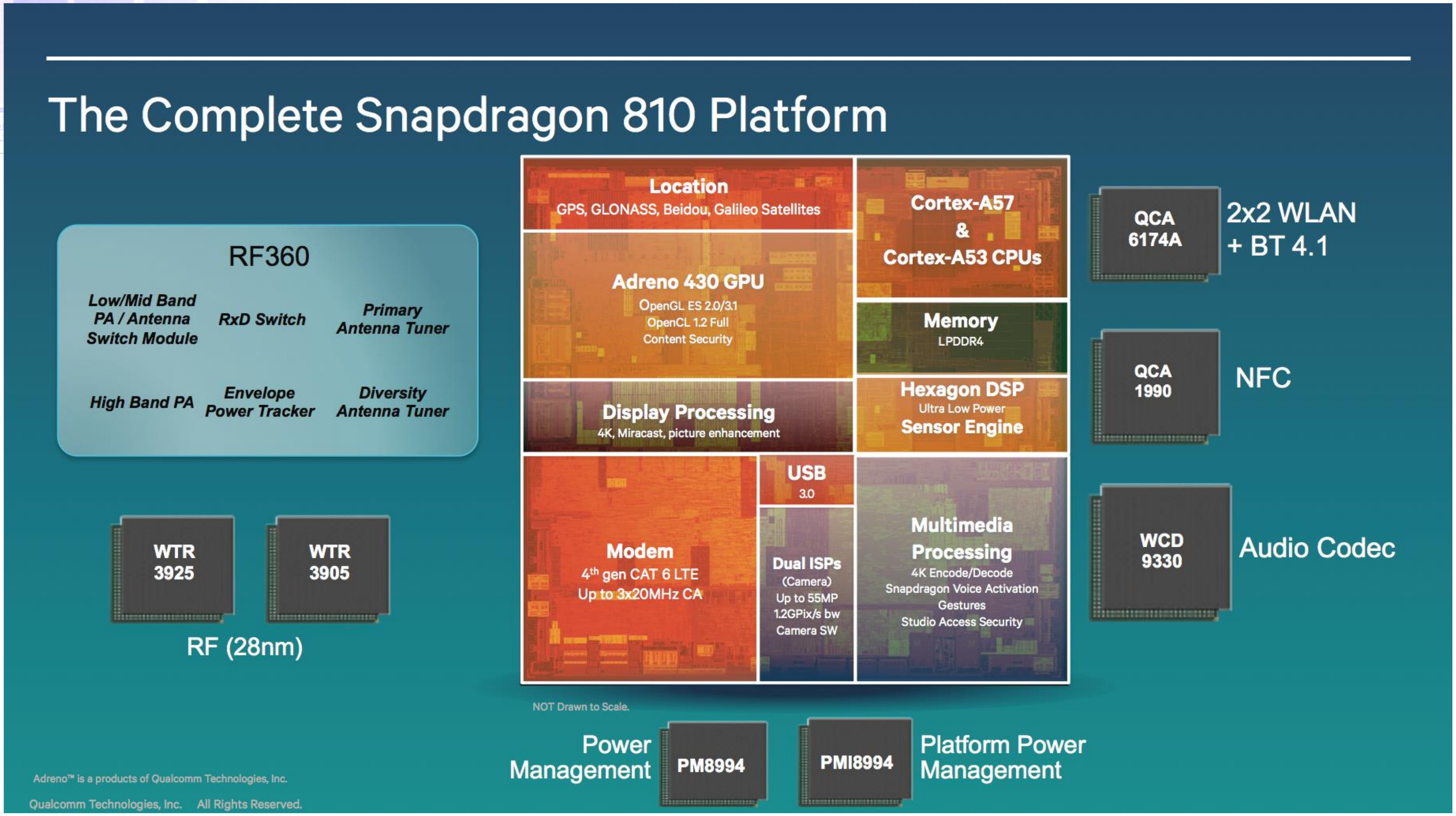
T. Kamei et al., “A resume-standby application processor for 3G cellular phones,” *ISSCC Dig. Tech. Papers*, pp. 336—337, Feb., 2004.

SoC Example: Smartphone Processor



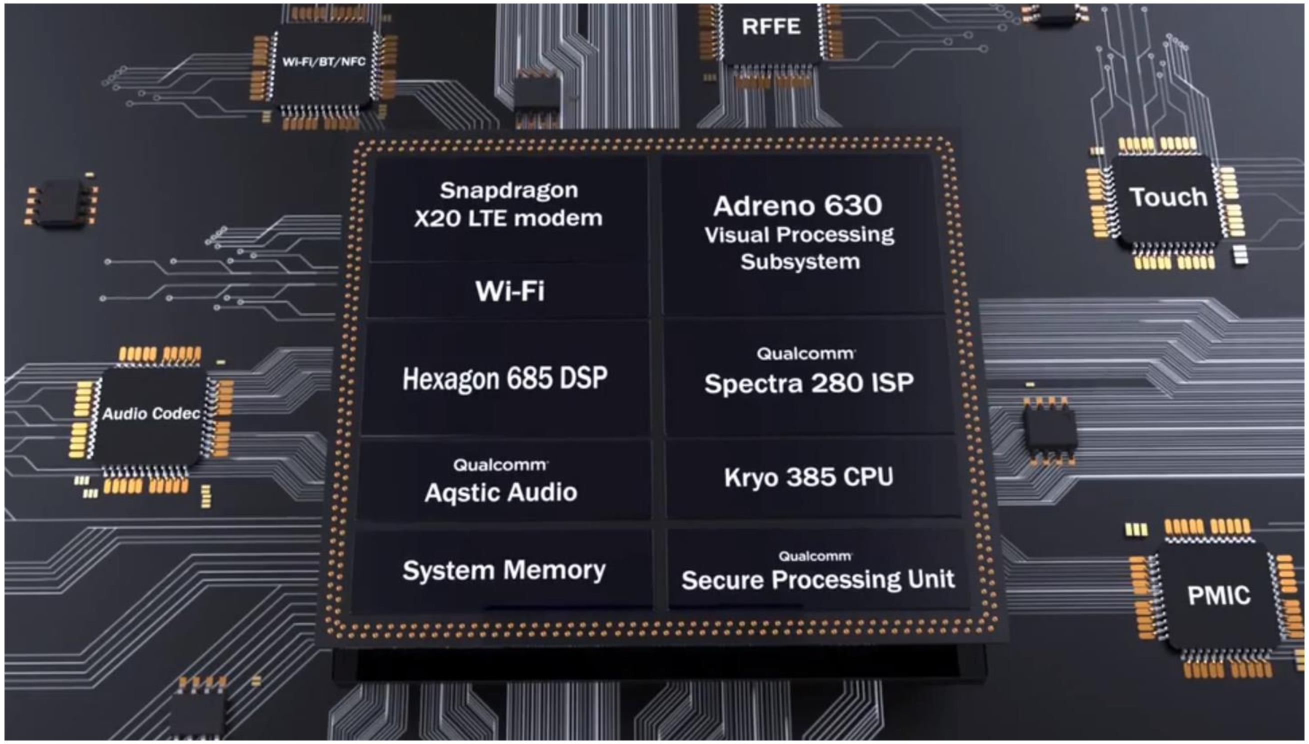
http://event.mediatek.com/_en_octacore/index.html

SoC Example: Smartphone Processor

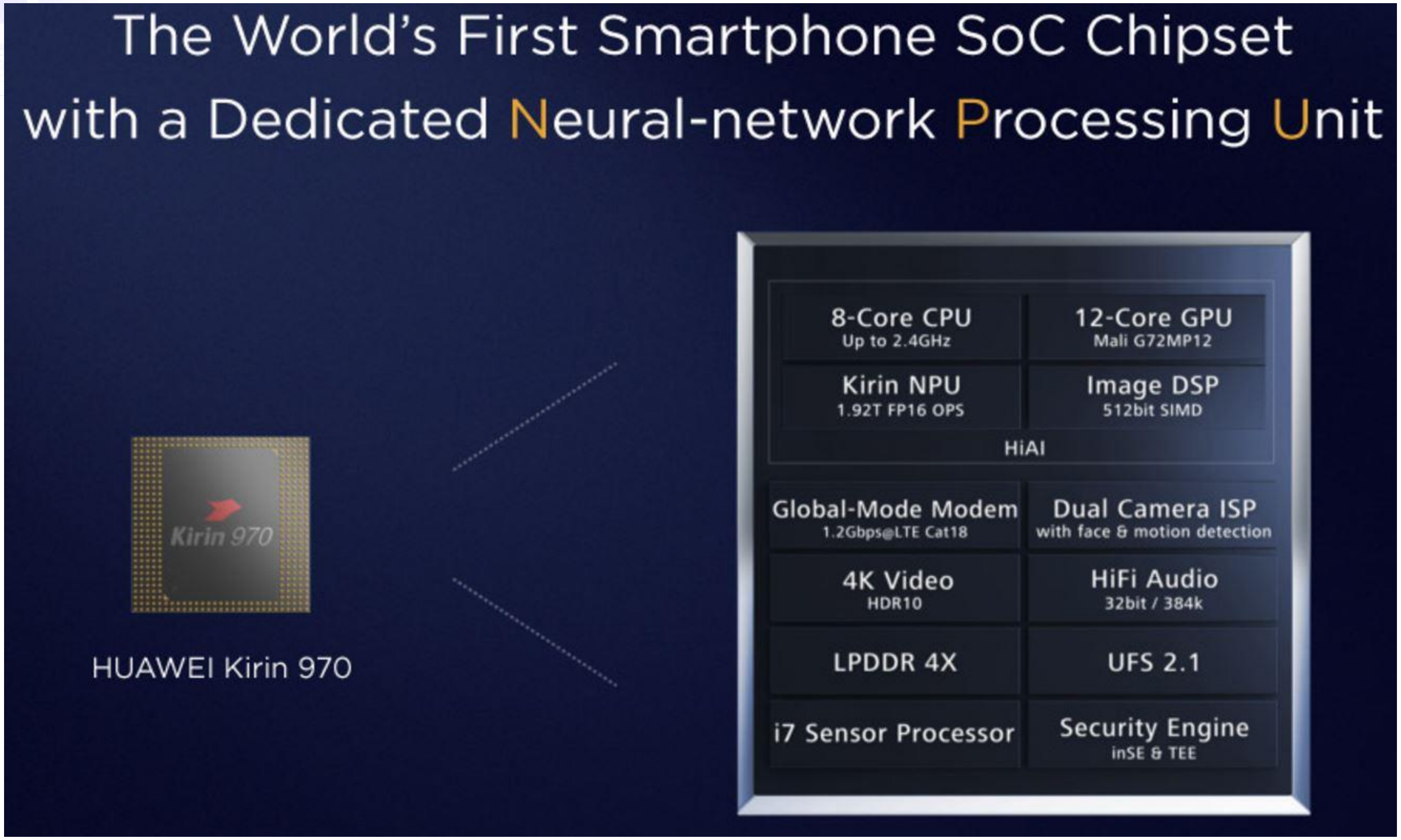


SoC Example: Smartphone Processor

- Snapdragon 845



SoC Example: Smartphone Processor



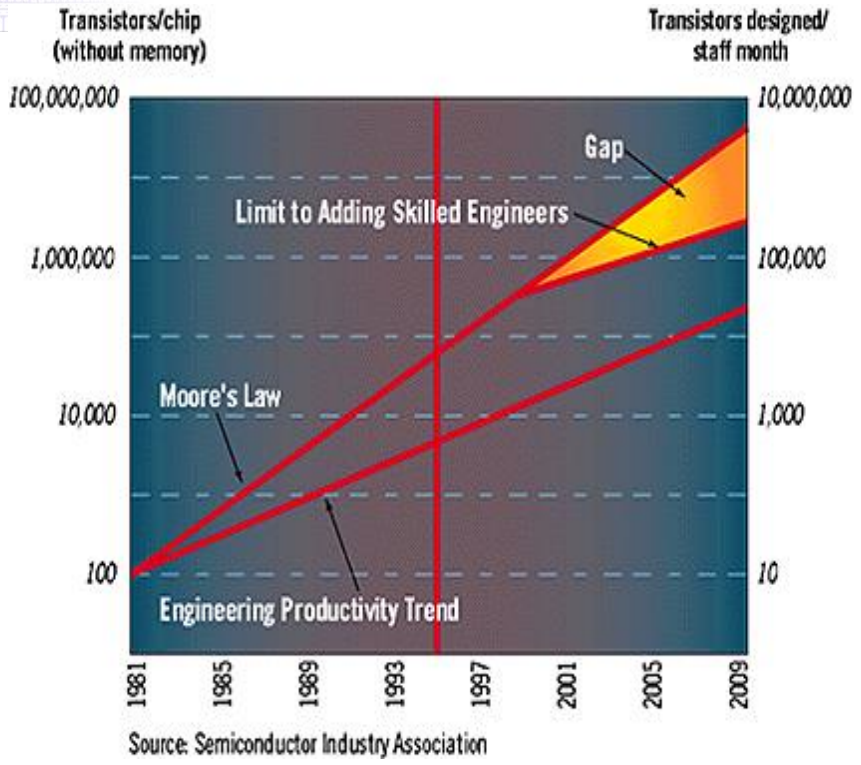
Outline

- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

SoC Dilemmas

- While SoC complexity is increasing, the time to market of consumer products is decreasing.
- IC designer lacks expertise of system developers.
- How to integration of internal virtual components (VC) and external VC?

Engineering Productivity Gap



- Engineering productivity has not been keeping up with silicon gate capacity for several years.
- Companies have been using <u>larger design teams, making engineers work longer hours</u>, etc., but clearly the limit is being reached.

Challenges

- Interoperability and Integration
  - IPs (Intellectual properties) present a multitude of interoperability and integration challenges. System-Level Integration
  - IPs may come in several forms: Hard, Soft, Firm
  - Common interface between blocks?

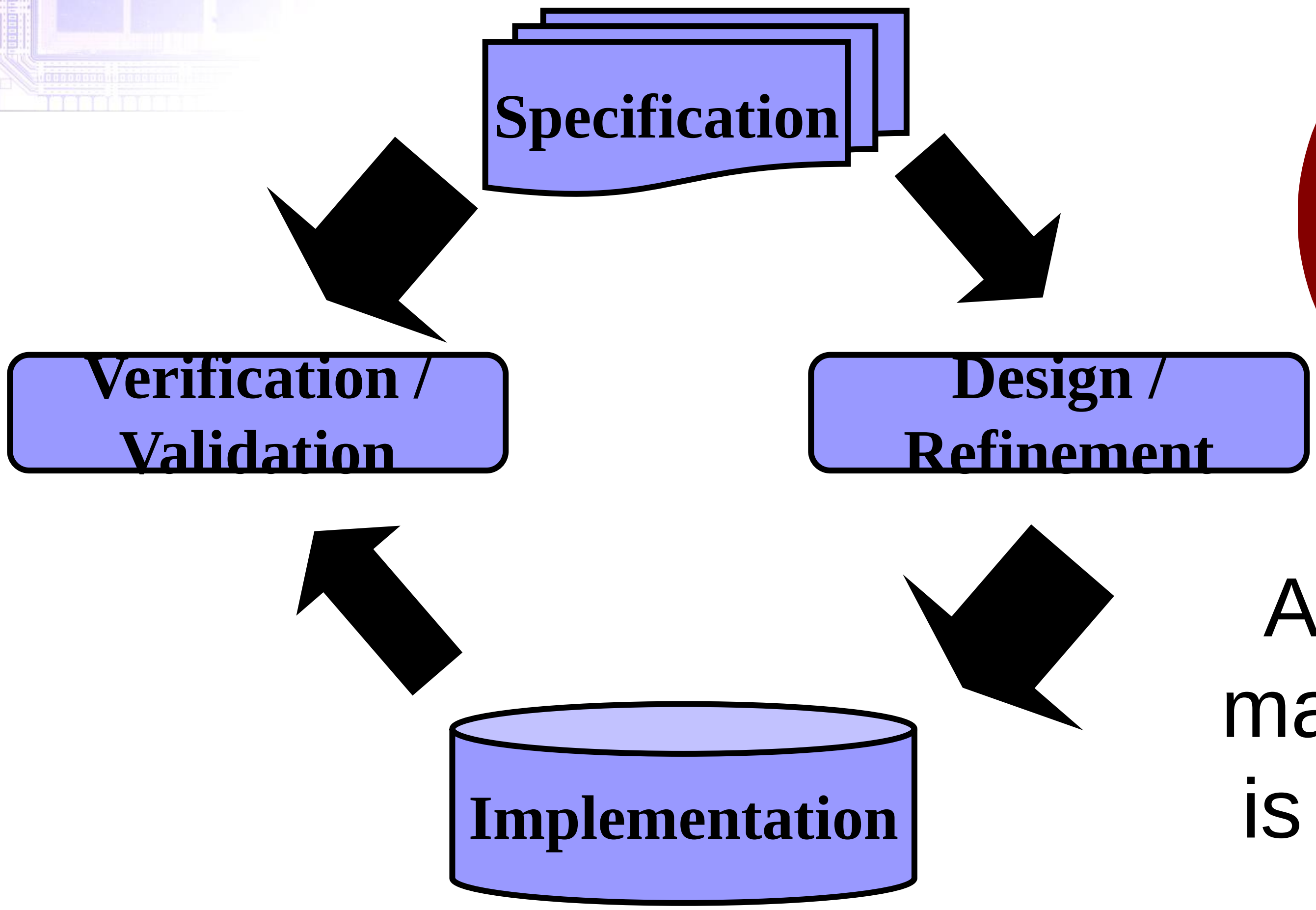
Challenges (cont.)

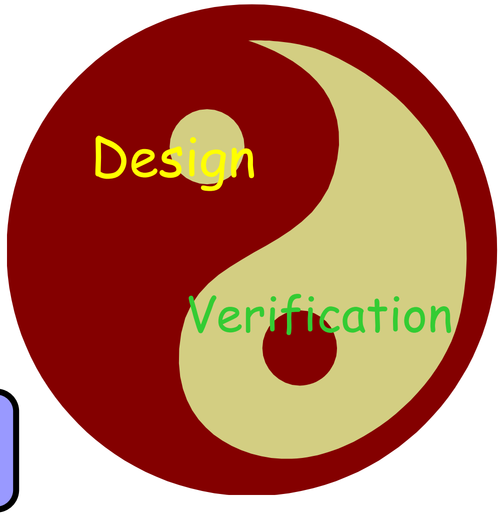
- EDA Tool Interoperability
  - These data formats may or may not be compatible.
  - Standardizing these diverse data formats.
- Testing an SoC
  - An SoC’s complexity requires extensive.
  - It’s necessary to test each VC separately.
- Process-Level Portability
  - Soft IP & Firm IP
  - Hard IP

Outline

- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

Design and Verification Step

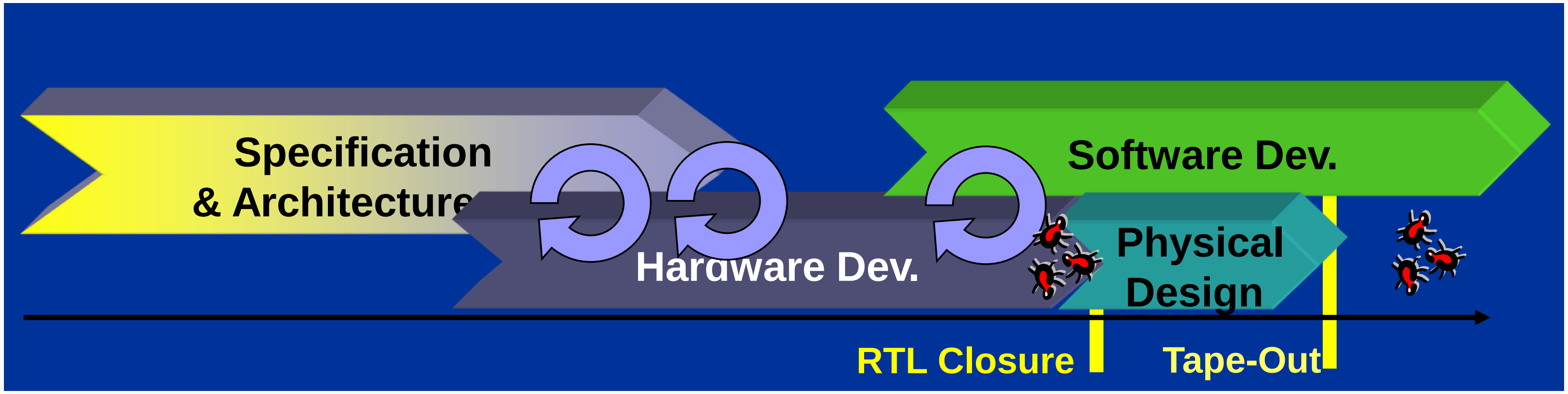




Actually, the major problem is Verification

Typical SOC design flow

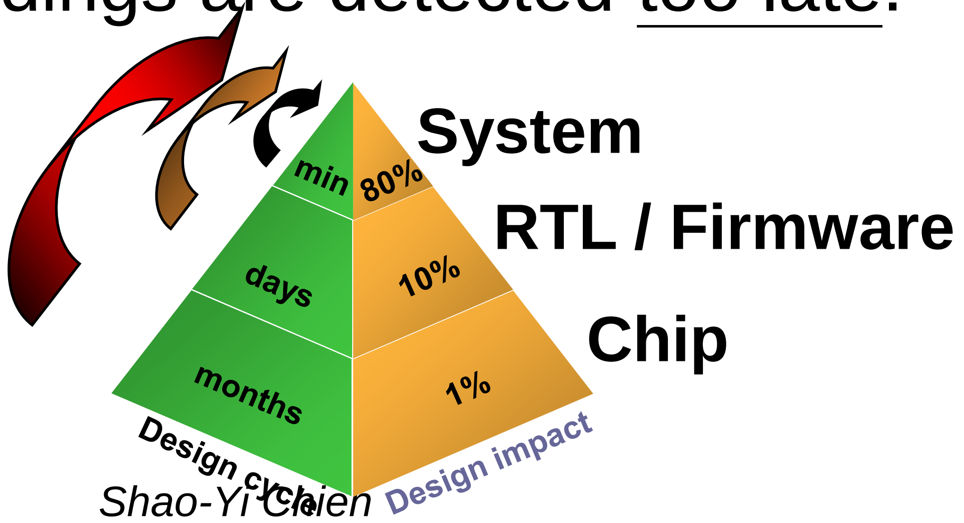
- Overlap in specification/architecture phase and RTL-design phase; multiple design changes
  - Architecture design done informally
- SW development starting late in the project



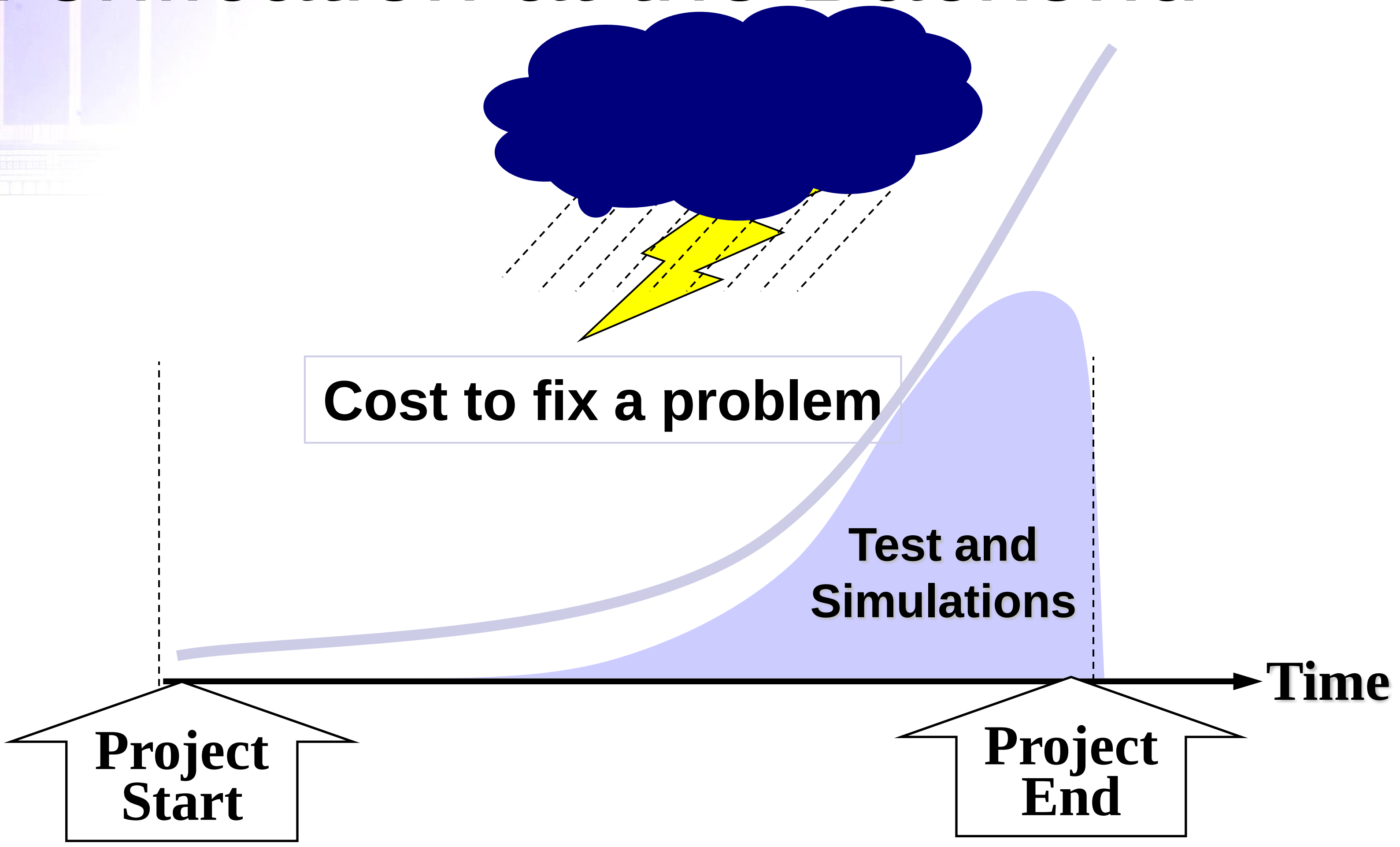
SoC Verification Gaps

- Different languages are spoken
  - At different levels of abstraction
  - By HW / SW / systems people

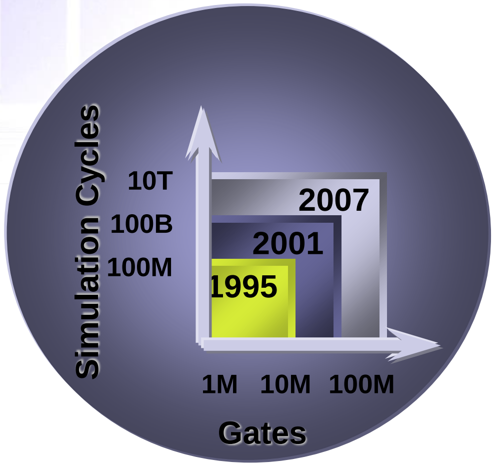
⇒ Problems, bottlenecks, and misunderstandings are detected too late.

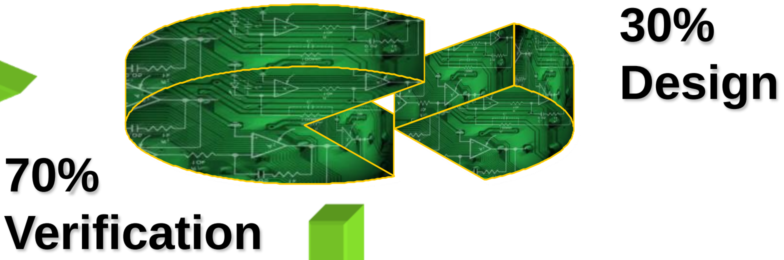


Verification at the Backend



SoC Verification Challenges



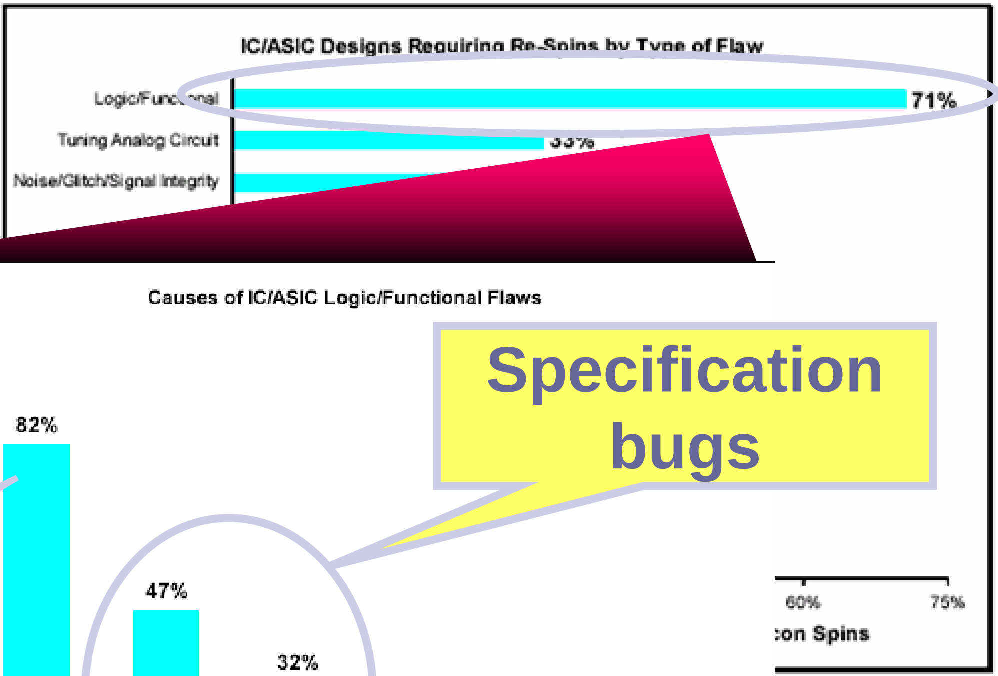


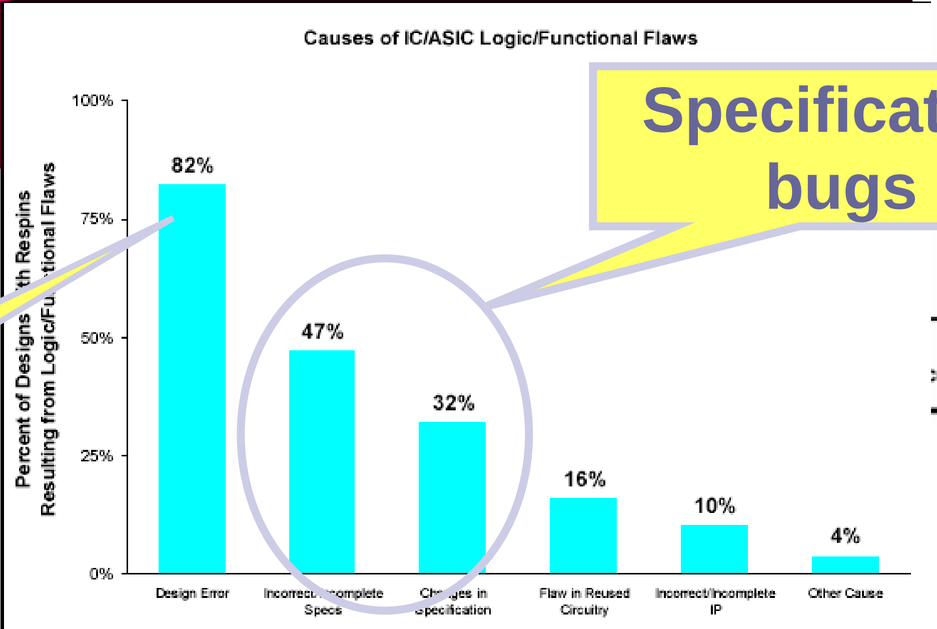
System Level Design Matters

65% in 2003

61%
of IC designs require one or more re-spins

Source:
2002 Collett International





Specification bugs

RTL bugs

Specification and RTL bugs cause re-spins!

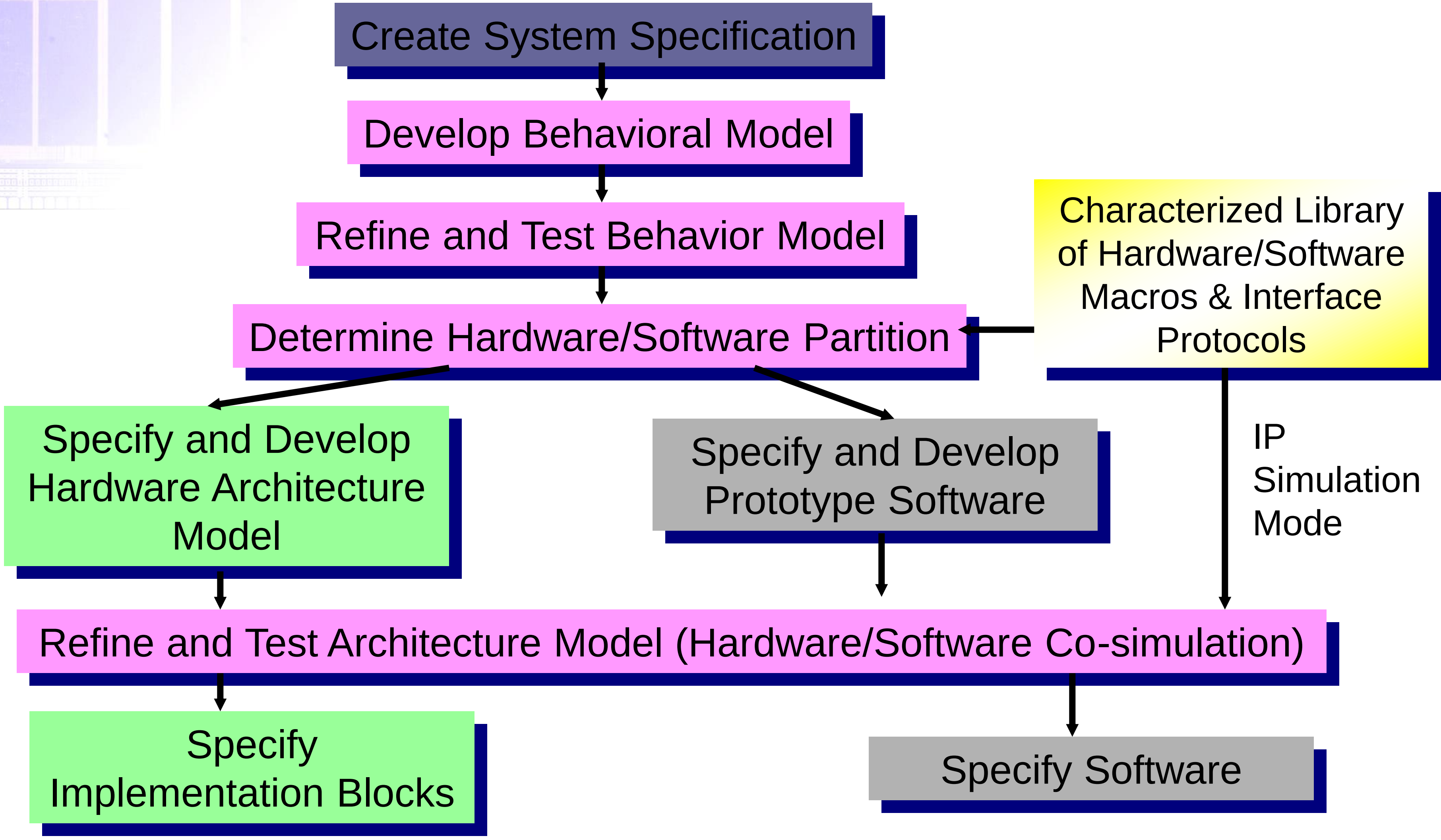
SoC Design Methodologies

System-Level IC Architecture	***IP Sourcing***	***IP Integration***	***Chip Implementation***	***Chip Fabrication***
•System Architecture **•Chip Architecture** **•Technology Selection** **•Algorithm Develop**	**•In-house IP** + **•3rd party IP** **-Selection** **-Qualification** **-Licensing**	**•Digital logic** + **•Mixed-signal** •Embedded Memory •Embedded Micro's	**•FPGA** **•Gate array** **•Standard cells** **•Megacell library** **•Datapath compiler** **•Memory compiler** + **•Hand-crafted** **•In-house tools**	**•3rd party foundry services**

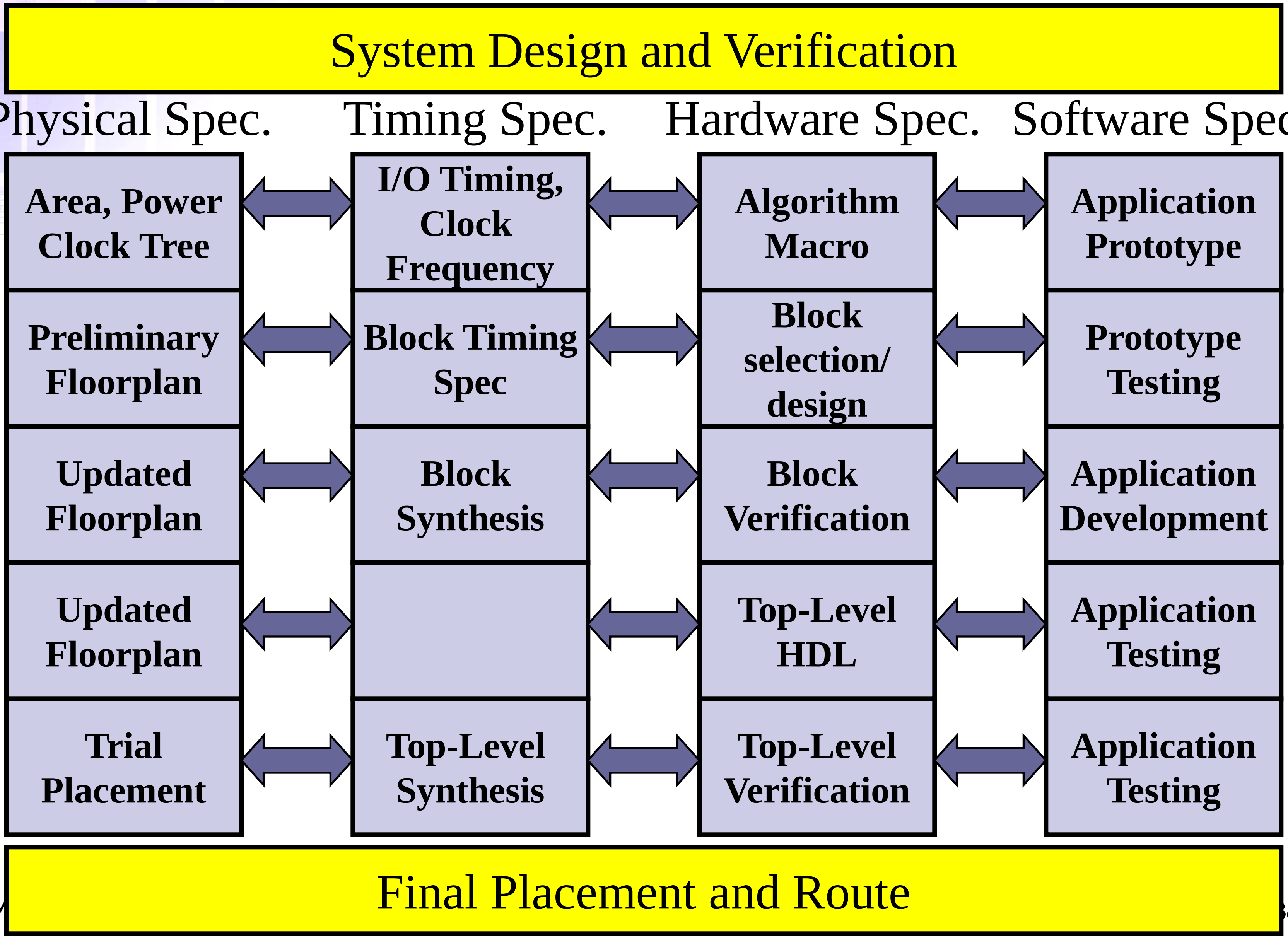
Note:Shaded area is the conventional ASIC development process

(Dr. H. D. Lin in 8th VLSI/CAD workshop)

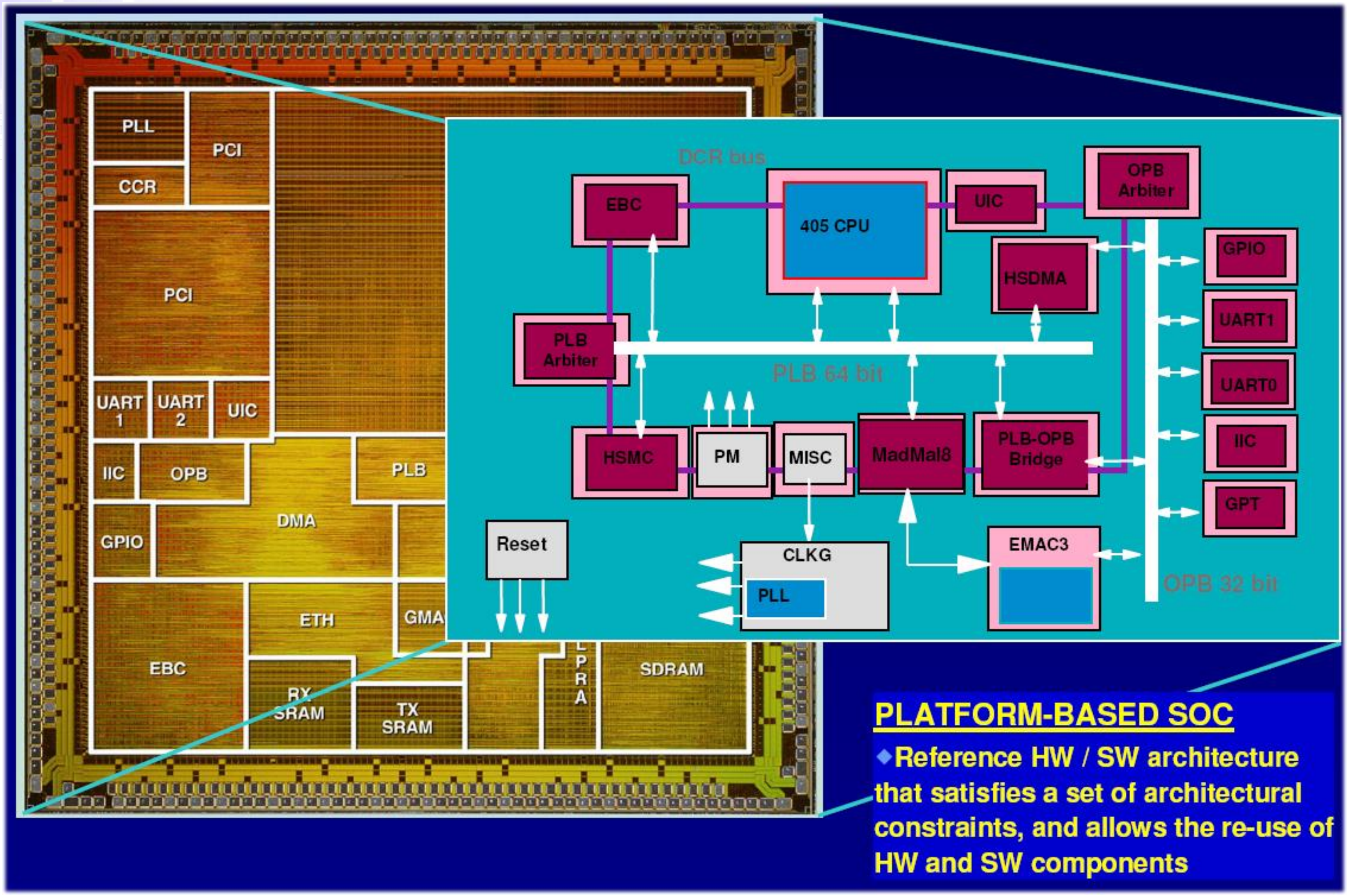
Top-Down Design Flow (RMM)


Create System Specification
Develop Behavioral Model
Refine and Test Behavior Model
Determine Hardware/Software Partition
Characterized Library of Hardware/Software Macros & Interface Protocols
Specify and Develop Hardware Architecture Model
Specify and Develop Prototype Software
IP Simulation Mode
Refine and Test Architecture Model (Hardware/Software Co-simulation)
Specify Implementation Blocks
Specify Software

Spiral SOC Design Flow (RMM)



Platform Based Design



PLATFORM-BASED SOC

- Reference HW / SW architecture that satisfies a set of architectural constraints, and allows the re-use of HW and SW components

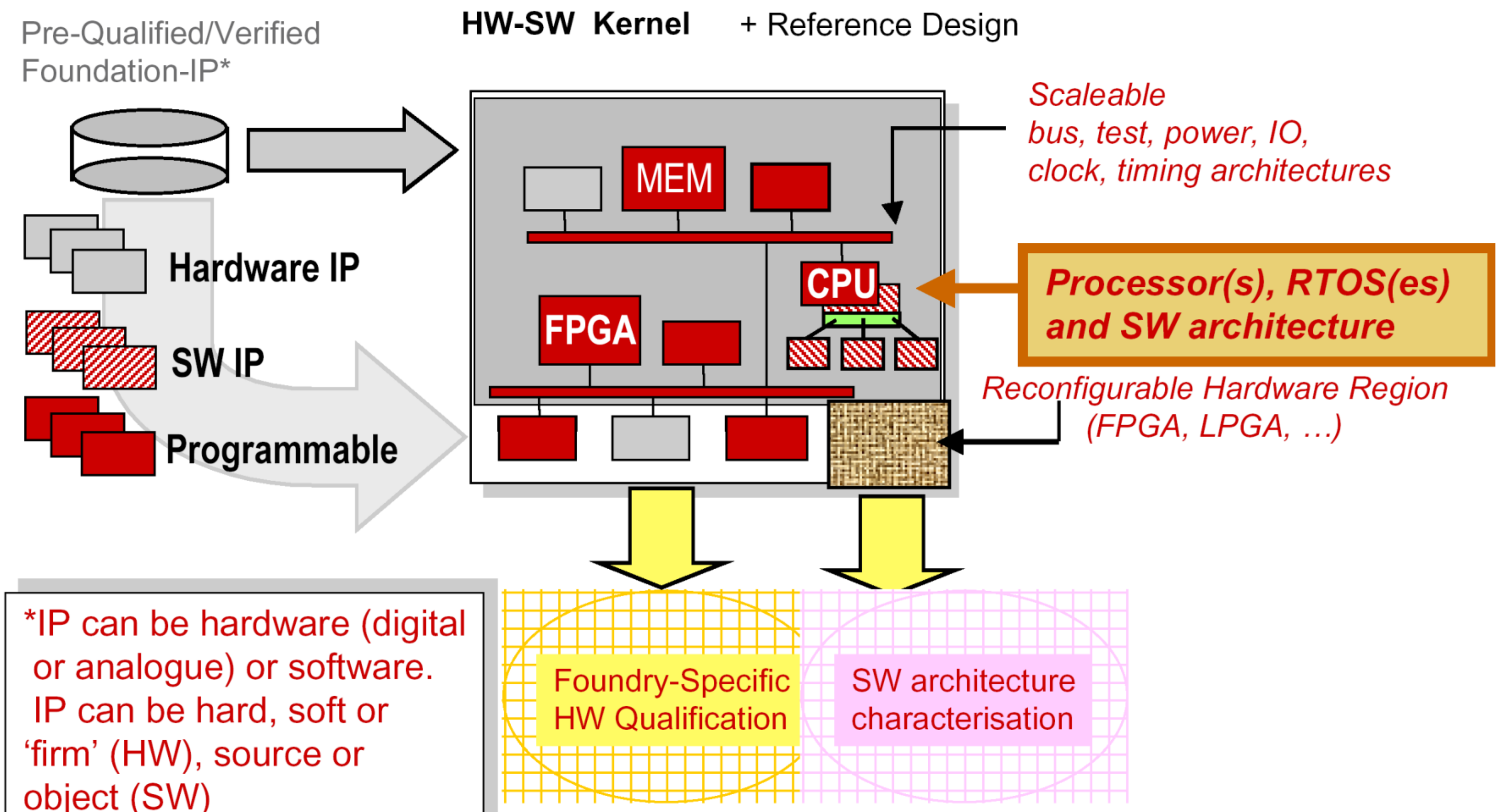
Platform Based Design

- Platform
  - **An integrated and managed set of common features**, upon which a set of products or **product family** can be built. A platform is a virtual component (VC).
- Platform-based design
  - An integration oriented design approach emphasizing **systematic reuse**, for developing complex products based upon platforms and compatible hardware and software VCs, intended **to reduce development risks, costs, and time to market**.

Platform Based Design

- More precise definition of platform-based design
  - An organized method to reduce the time required and risk involved in designing and verifying a complex SoC, by heavy reuse of combinations of hardware and software IP. Rather than looking at IP reuse in a block by block manner, platform-based design aggregates groups of components into a reusable platform architecture.
- System platform
  - A coordinated family of hardware-software architectures, satisfying a set of architectural constraints that are imposed to allow the reuse of hardware and software components

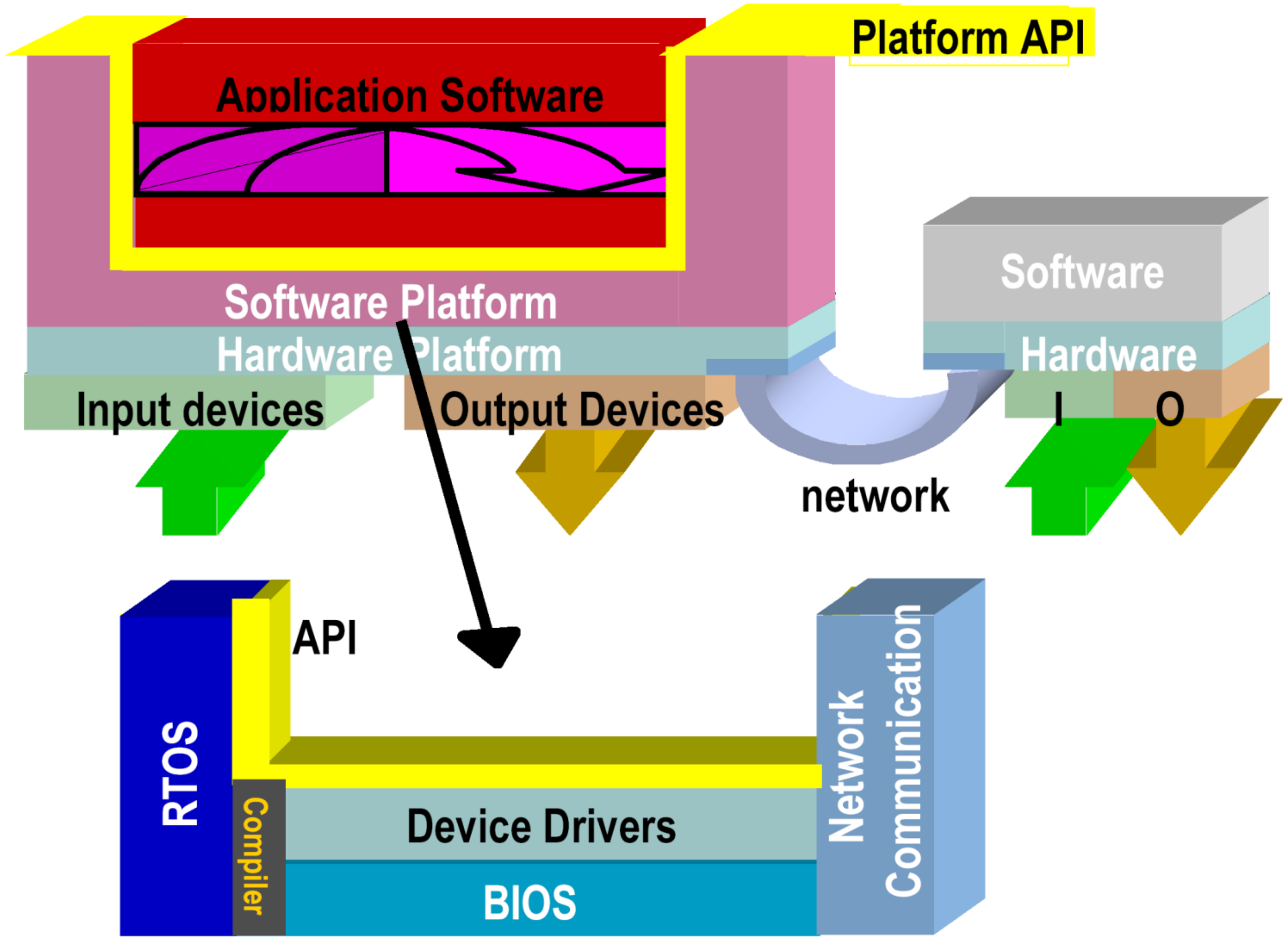
A Hardware-centric View of a Platform


Pre-Qualified/Verified Foundation-IP*
HW-SW Kernel
+ Reference Design
Scaleable bus, test, power, IO, clock, timing architectures
MEM
CPU
FPGA
Hardware IP
SW IP
Programmable
Processor(s), RTOS(es) and SW architecture
Reconfigurable Hardware Region (FPGA, LPGA, …)
Foundry-Specific HW Qualification
SW architecture characterisation

*IP can be hardware (digital or analogue) or software. IP can be hard, soft or 'firm' (HW), source or object (SW)

Source: Grant Martin and Henry Chang, ISQED 2002 Tutorial

A Software-centric View of a Platform



Source: Grant Martin and Henry Chang, ISQED 2002 Tutorial

Other Design Techniques/Problems

- Hardware/software partition
- Hardware/software co-design
- Hardware/software co-verification

- The EDA tool?

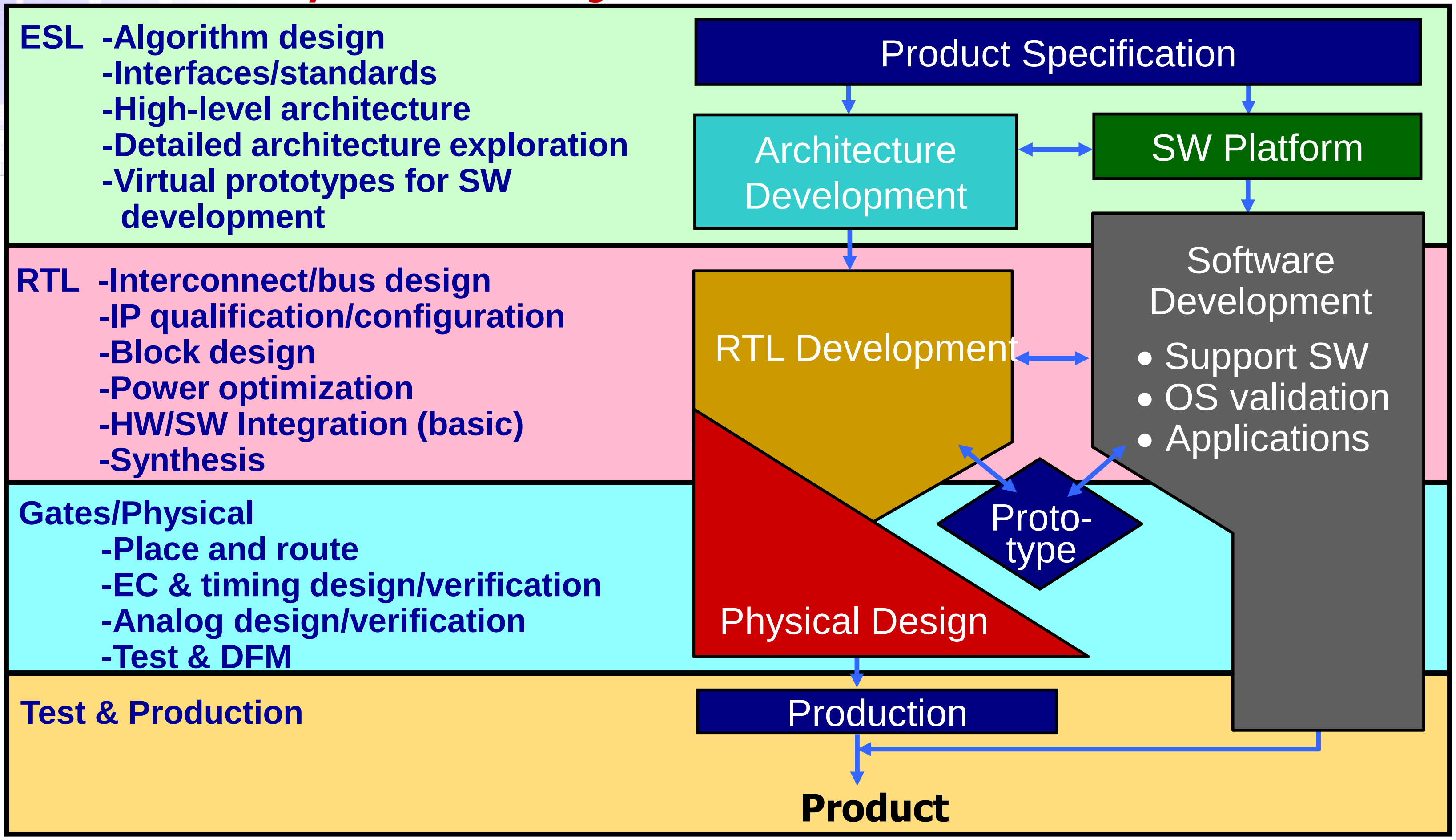
Outline

Emerging SoC Design Flow (1/2)

- The design methodologies developed for earlier SoC technology are inadequate to the task of designing a multiprocessor SoC
  - Electronic system-level (ESL) design methodology has been devised to solve these problems
- Virtual Prototype
  - A high-speed (20MHz or more) functional model of the target chip
  - Can quickly assemble, simulate, and analyze alternative architectures
  - Allows software development to start many months before a hardware prototype is available

Emerging SoC Design Flow (2/2)

ESL : Electronic System Level Design



Source: Synopsys, Inc.

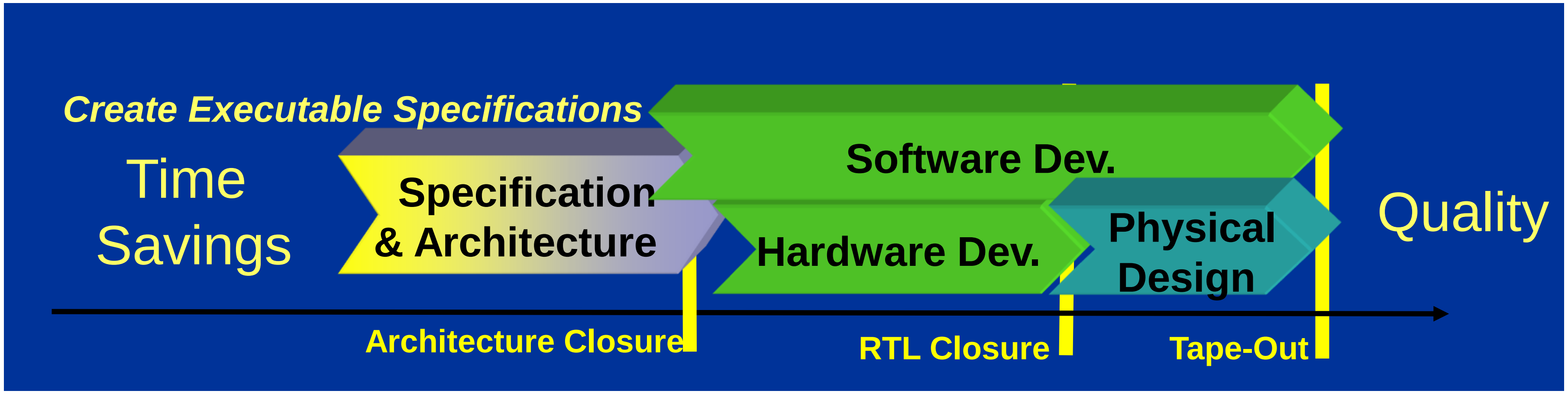
Electronic System-Level (ESL) Design

- A set of methodologies that enables SoC engineers to efficiently develop, optimize and verify complex system architectures and embedded software

- The foundation for the continuously verifying downstream register-transfer level (RTL) implementation and subsequent software development

ESL: New SOC Design Flow

- Architecture closure
  - Achieve a reduction # of RTL iterations
  - Can perform concurrent HW and SW design
  - Shorten the time it takes to get to golden RTL



Architectural Closure

Model the entire system (HW & SW) to verify that it meets the performance goals optimally

- Validate the architecture
  - Eliminate bottlenecks in Bus transactions
  - Refine data buffer structure/management
  - Close on HW/SW partitioning
- Perform software-based testing
  - Verify system setup, peripheral drivers and key application SW features
  - Optimize timing-critical tasks of the embedded software

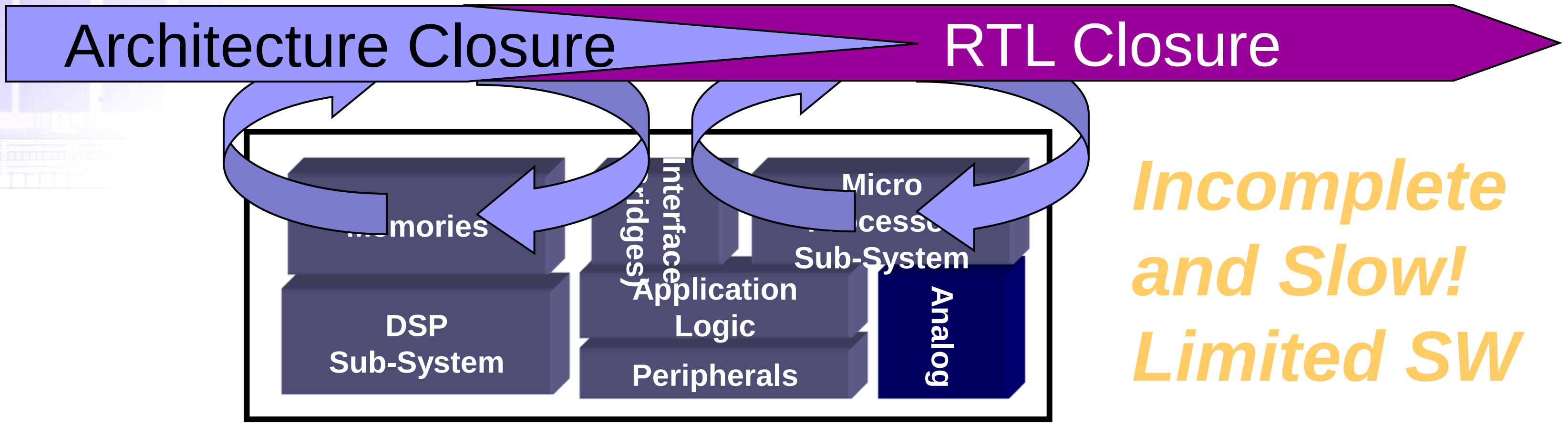
RTL Closure

Goal: Implement and verify the architecture in RTL

- Individual block (IP) level
  - Implement/synthesize RTL blocks or
  - Import (& re-validate) design IP
  - "Prove" block-level functionality and performance
  - Check conformance to specifications/standards
- Full chip level
  - Resolve micro-architecture corner cases (clock domains, FIFOs, handshakes, split Bus transactions)
  - Integrate imported IP, show chip-level integrity
  - Perform software execution (reset……)

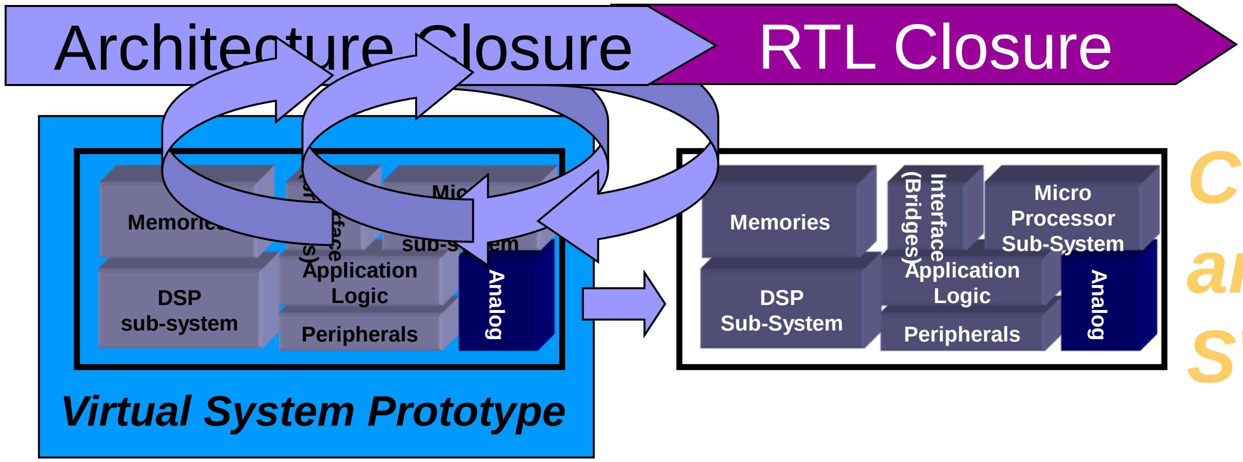
SOC Design Flows

Typical Flow: ***Step 1 and 2 performed on RTL model***



Incomplete and Slow! Limited SW

New Flow: ***Step 1 on transaction level, step 2 on RTL model***

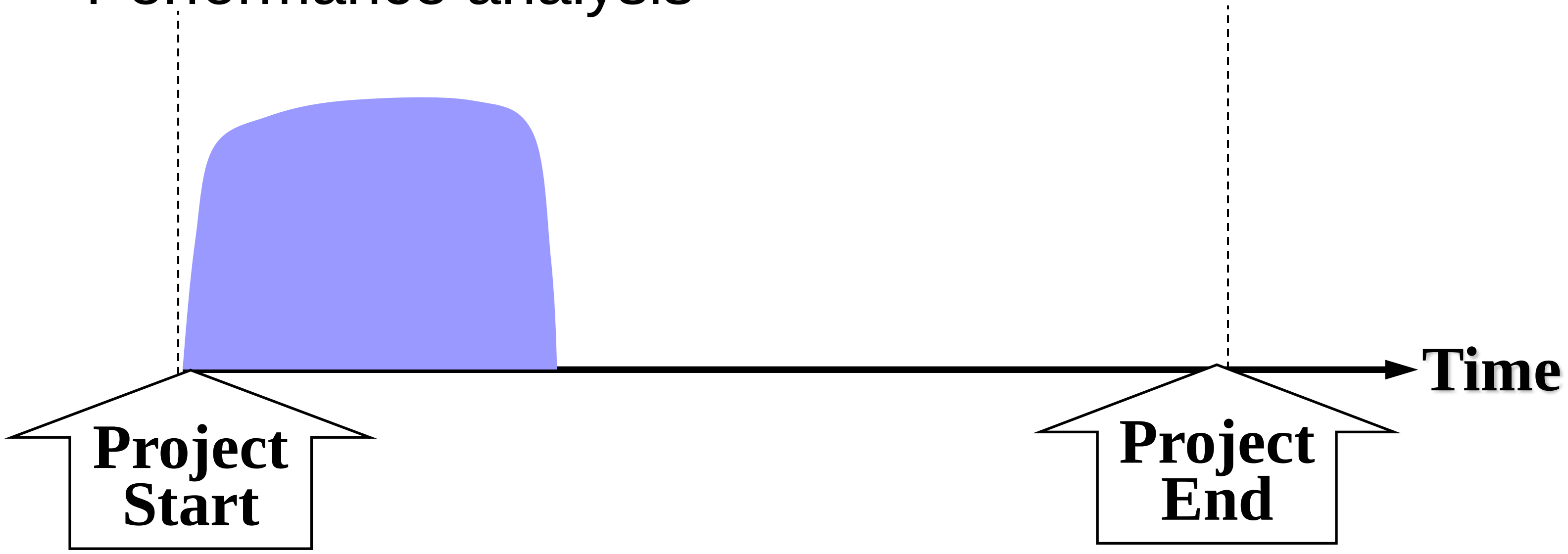


Complete and Fast! SW Early

Continuous Verification (I)

- High Level Analysis
  - Functional verification
  - Architecture exploration
  - Performance analysis

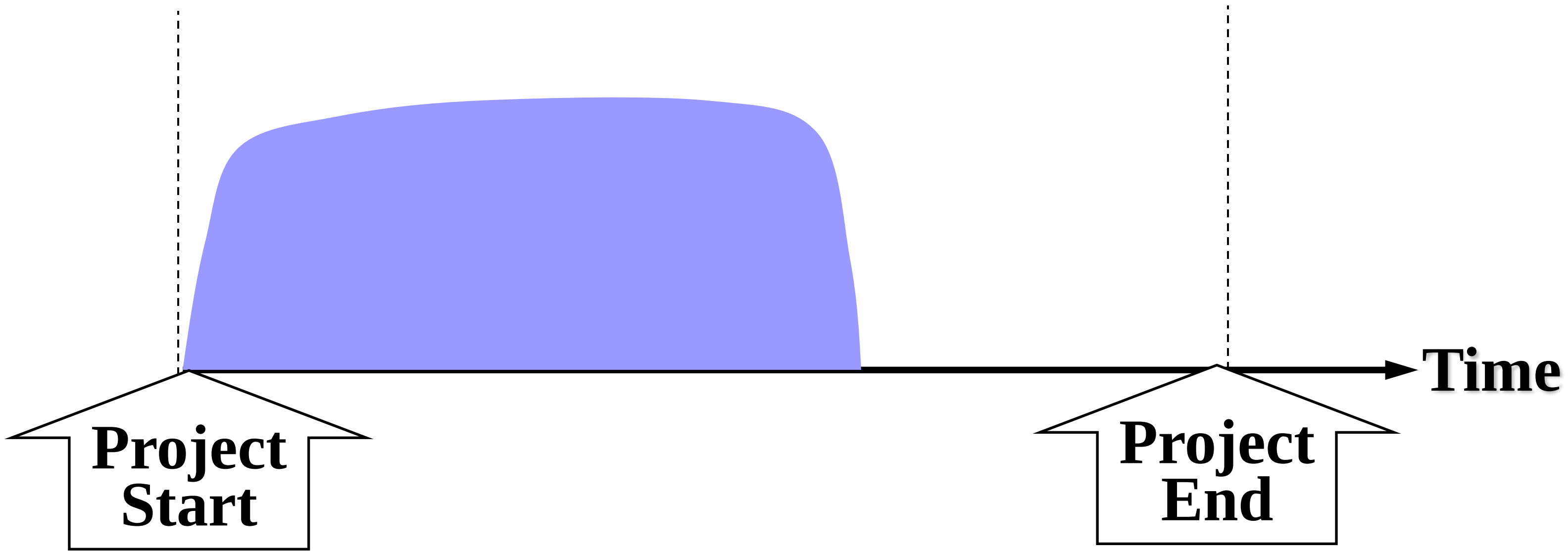
- Executable specification
- High-level testbench
- SW development platform



Continuous Verification (II)

- "Mixed" Level Analysis
  - Functional verification
  - Architecture validation
  - Performance validation

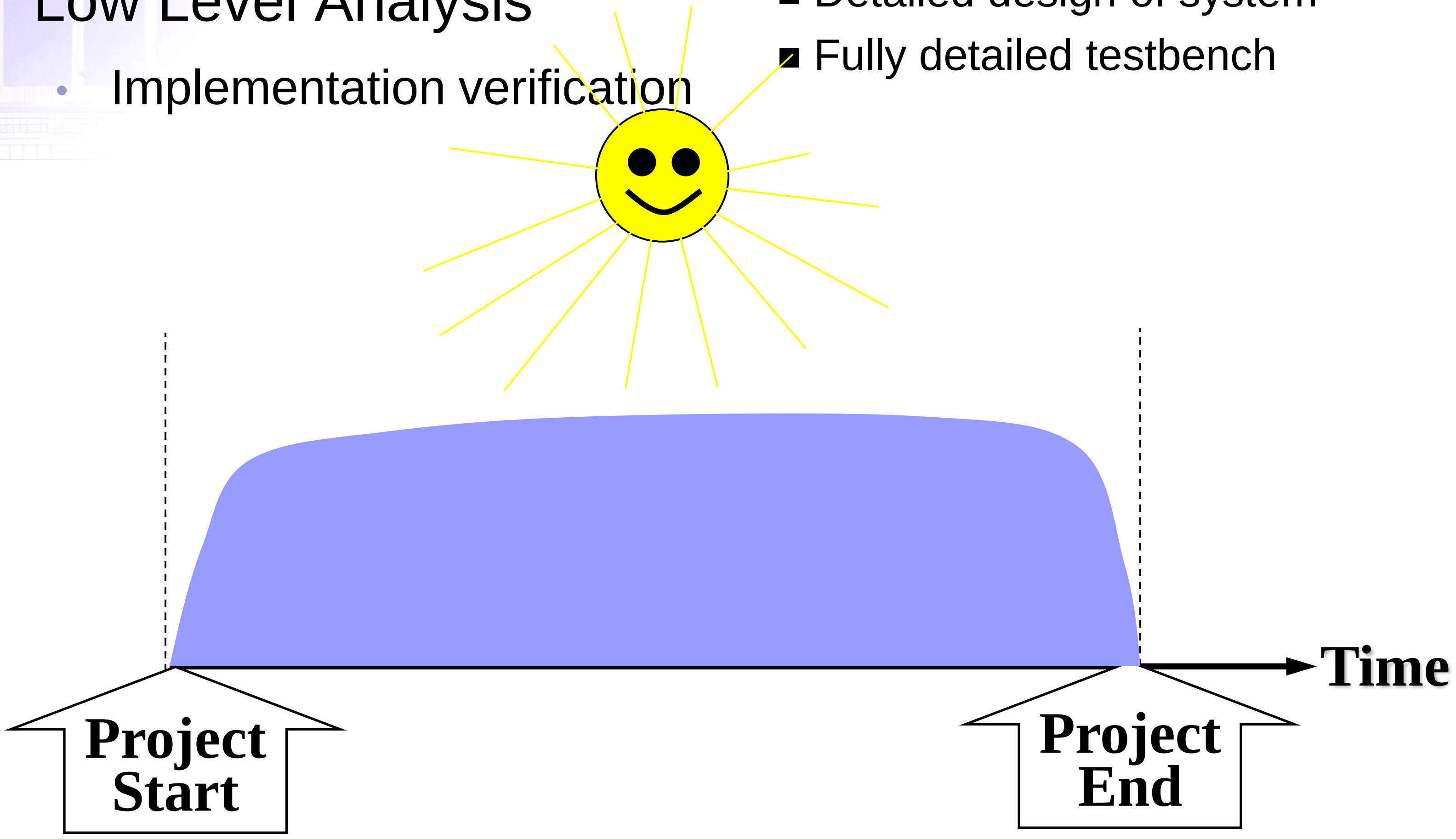
- Re-used IP
- Detailed design of components/subsystems
- More detailed testbench



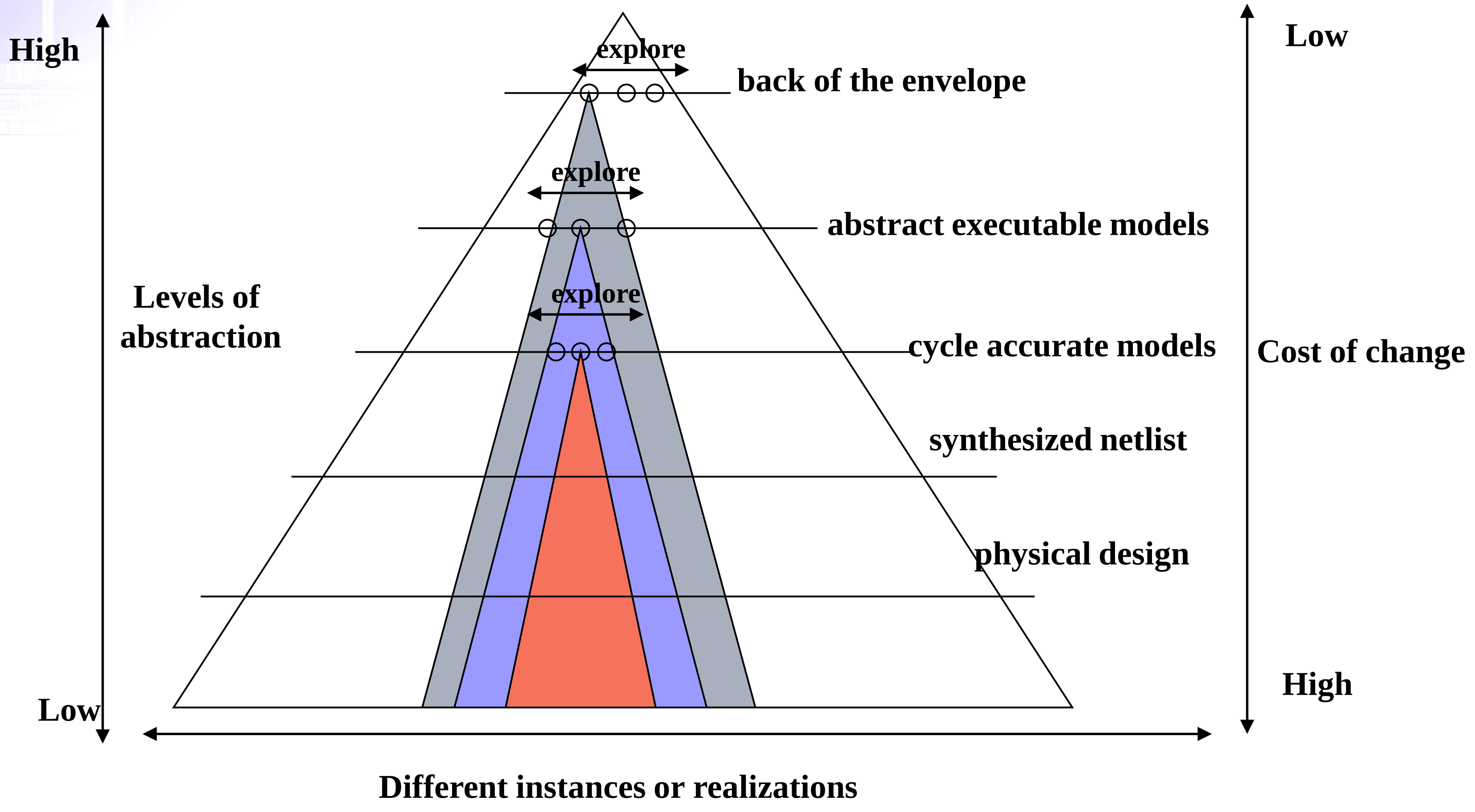
Continuous Verification (III)

- Low Level Analysis
  - Implementation verification

- Detailed design of system
- Fully detailed testbench



Design Space Exploration



Outline

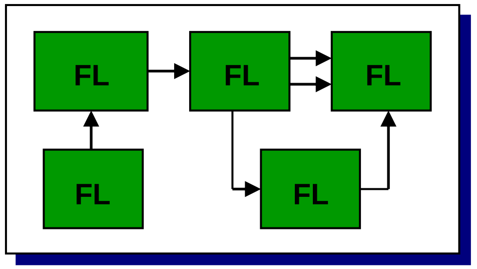
- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

Modeling issues

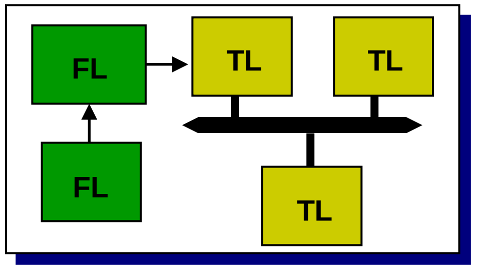
- System-level design tools will be integrated into the new SoC design flow
  - Also called as Electronics System Level (ESL) tools
- Have benefits in system verification and hardware-software co-design
- Good modeling is the key for successful system-level design

Levels of Abstraction

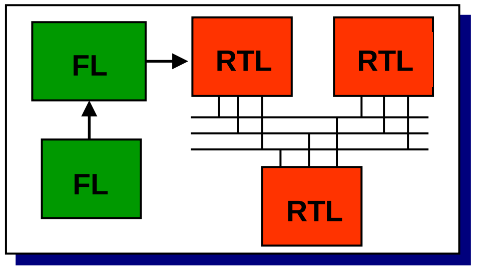
- Functional Level
  - Algorithm optimization
  - Dropped calls/bit error rate



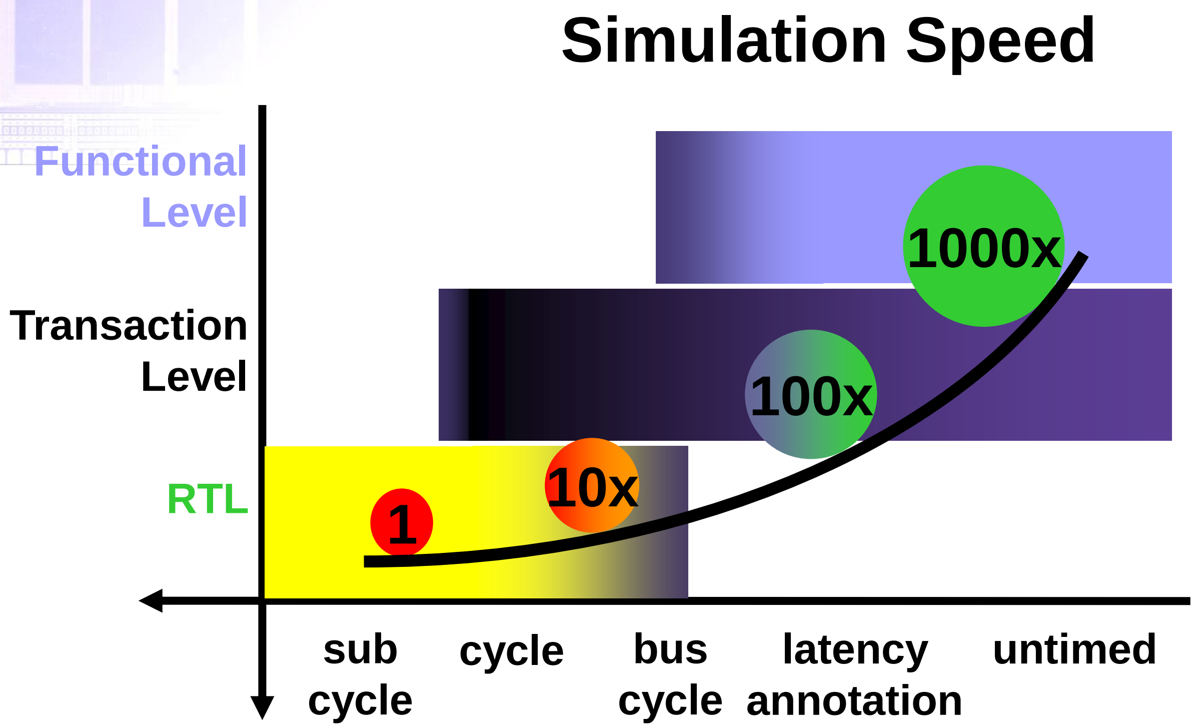
- Transaction Level
  - Architecture closure
  - Software verification
  - Bus bandwidth / cache size



- RT-Level
  - Detailed hardware design
  - handshake / timing issues



SoC Verification Issues



Audio/video quality ?
bit error rate ?

bus bandwidth ?
cache size ?

handshake ?
reset ?

Simulation speed requirements :: 100-1000x

Functional Level Modeling

High Performance

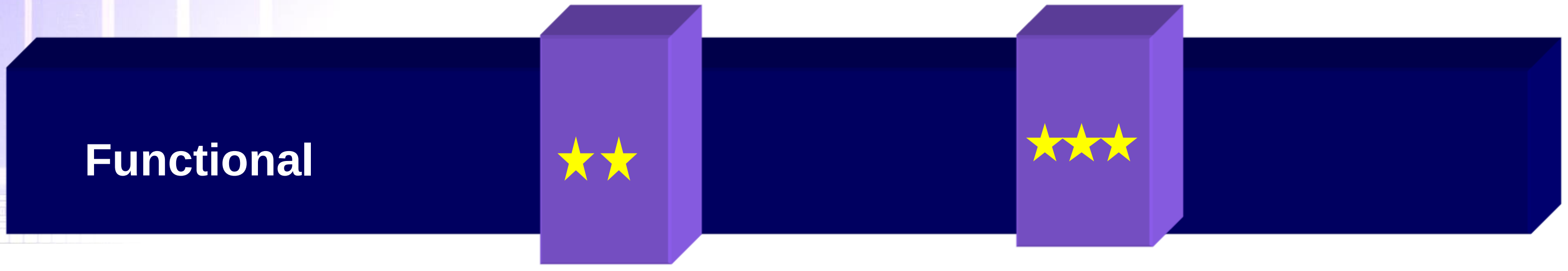
Functionality	Yes
Cycle Accurate	No
Timing	No
Pin Accuracy	No
Communication	Point to Point
Channels	FIFO
Parameters	Yes

Functional Modeling - Benefits

- High Performance
  - potential for 1000x speed over RTL
- Model the “complete” system and environment
  - provides a functional testbench that can be used during implementation
- System level analysis capabilities
- Libraries of standard protocols jumpstart modeling efforts
  - e.g., CDMA/Bluetooth – reference design kits

Languages

★★★ excellent
★★ good
★ ok



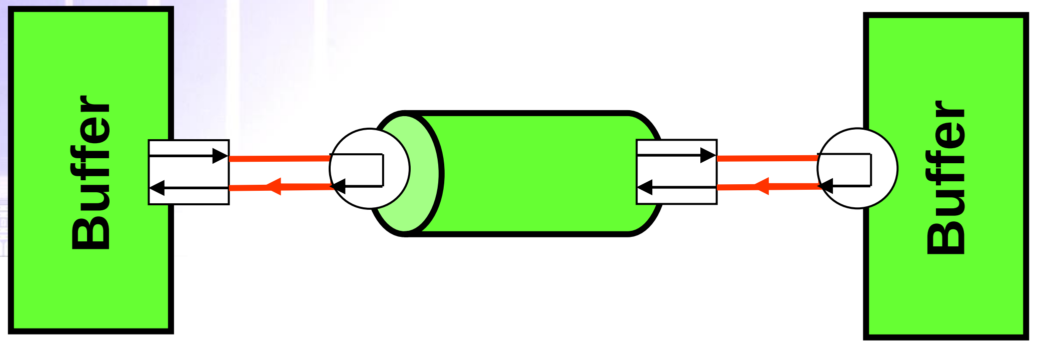
SystemC

Ptolemy
Matlab

Transaction Level (TL) Modeling

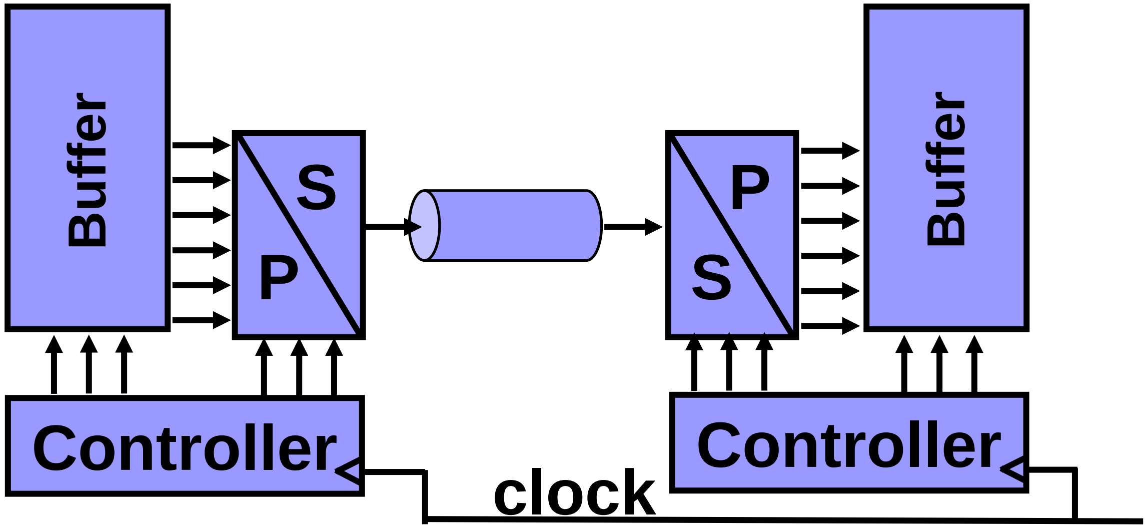
Functionality	Yes
Cycle Accurate	Not necessary
Timing	No
Pin Accuracy	No
Communication	Shared
Channel	User defineable
Parameters	Yes

TL Modeling



Transaction Level

```
write(ATM_cell..)
```



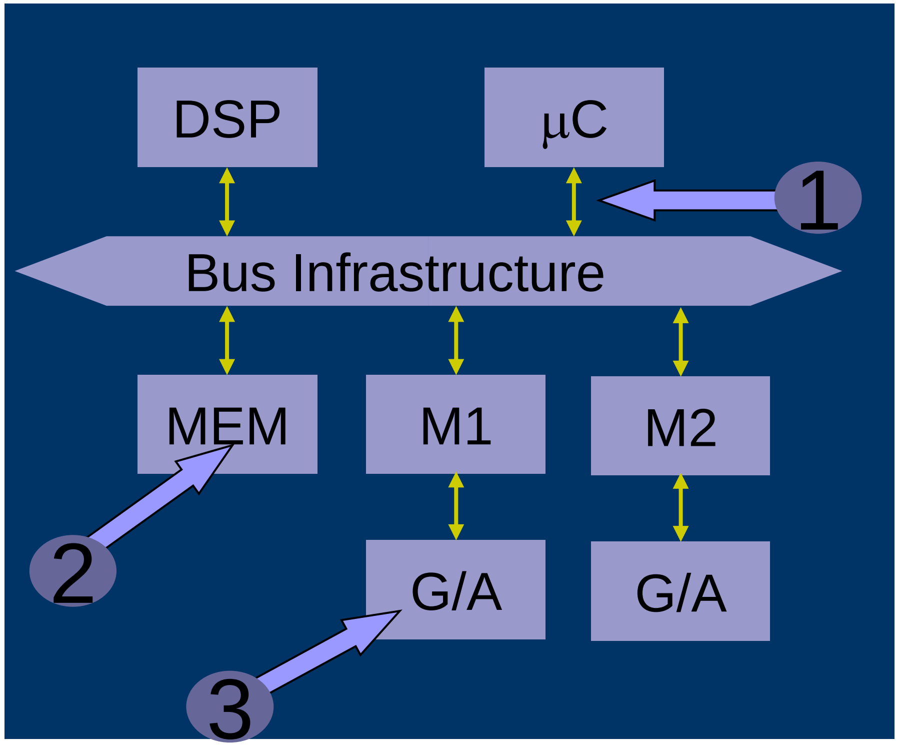
RT-Level

```
always(@posedge ..
    S1=R2;
    S2=R25;
```

One Transaction >>>>>>>>>> 1000 signals toggling 1000 times

TL Modeling

Transaction level modeling focuses on the communication between concurrent functional modules through the (on chip) bus infrastructure



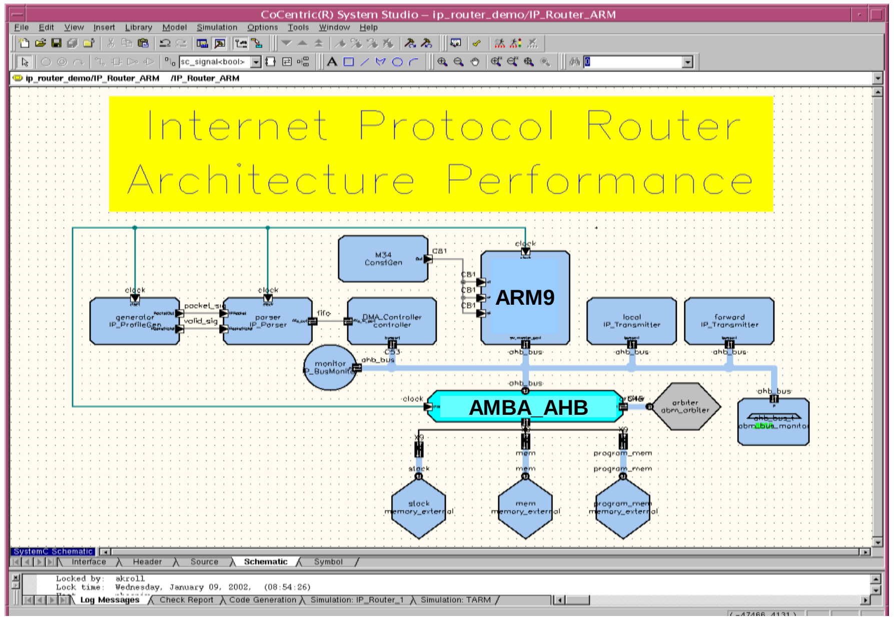
1. All modules have well defined procedural interfaces to communicate with other modules
2. Modules model the function and (context sensitive) latency between request/response
3. Sources and sinks model real world data rates
   - Processor, packet streams, etc.

Fast Architecture Verification

Design Capture
- **multiple levels of abstraction**
- **graphical, textual**

Debug of HW & SW
- **source code debug**
- **memories, buses, interrupts**

Performance Analysis
- **interactive traces and statistics**

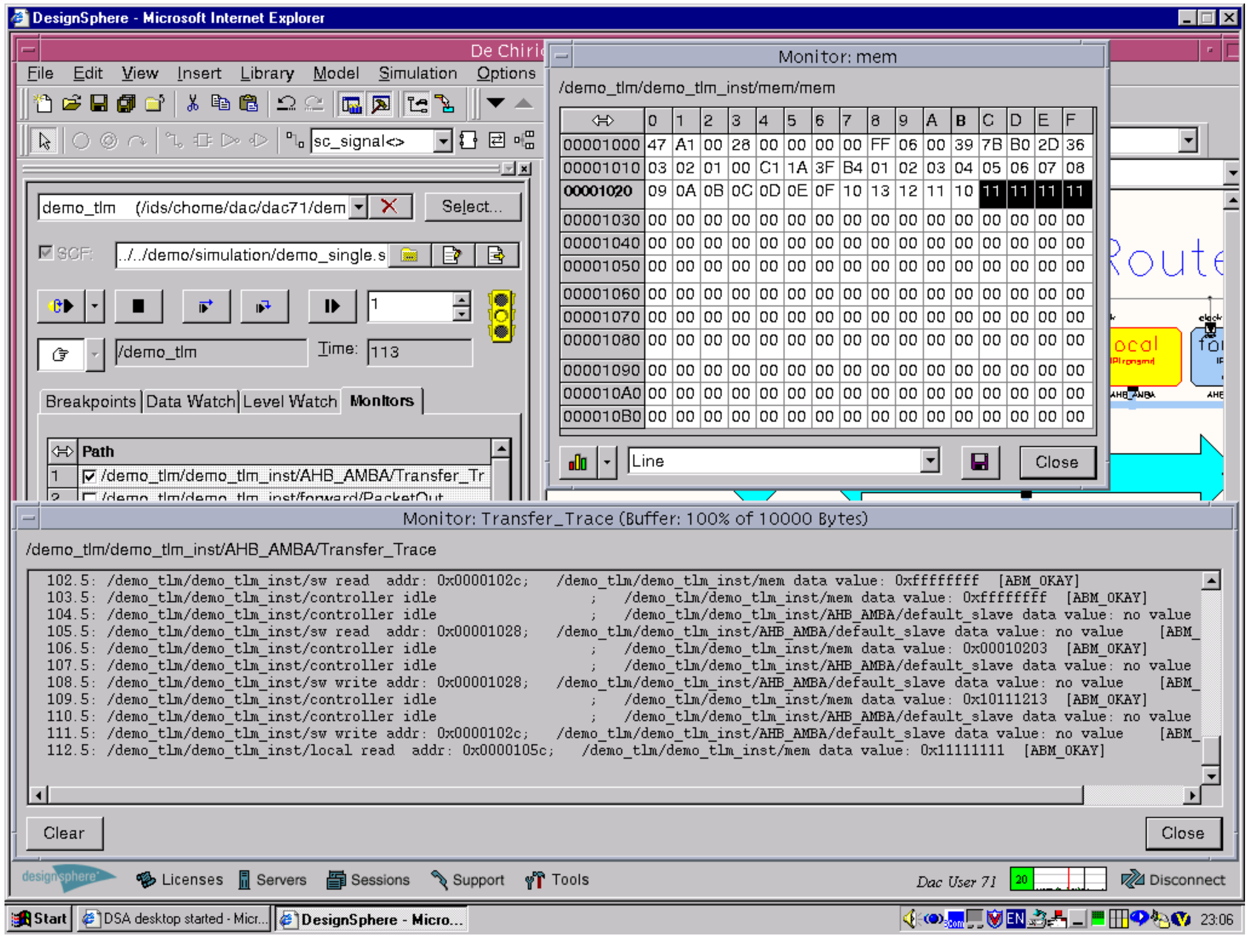


```
Locked by:  akroll
Lock time:  Wednesday, January 09, 2002,  (08:54:26)
```

Closing on the Architecture at the Transaction Level reduces Risk by 80%

TL Modeling

Bus and Memory Analysis


DesignSphere - Microsoft Internet Explorer
De Chiric
File Edit View Insert Library Model Simulation Options
Select...
Breakpoints Data Watch Level Watch Monitors
Path
Licenses Servers Sessions Support Tools
Dac User 71
Disconnect
Start
DSA desktop started - Micr...
DesignSphere - Micro...
23:06

demo_tlm (/ids/chome/dac/dac71/dem

SCF: ../../demo/simulation/demo_single.s

sc_signal<>

/demo_tlm Time: 113

1 /demo_tlm/demo_tlm_inst/AHB_AMBA/Transfer_Tr

2 /demo_tlm/demo_tlm_inst/forward/PacketOut

Monitor: mem

/demo_tlm/demo_tlm_inst/mem/mem

⇔	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00001000	47	A1	00	28	00	00	00	00	FF	06	00	39	7B	B0	2D	36
00001010	03	02	01	00	C1	1A	3F	B4	01	02	03	04	05	06	07	08
00001020	09	0A	0B	0C	0D	0E	0F	10	13	12	11	10	11	11	11	11
00001030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000010A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000010B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Line — Close

Monitor: Transfer_Trace (Buffer: 100% of 10000 Bytes)

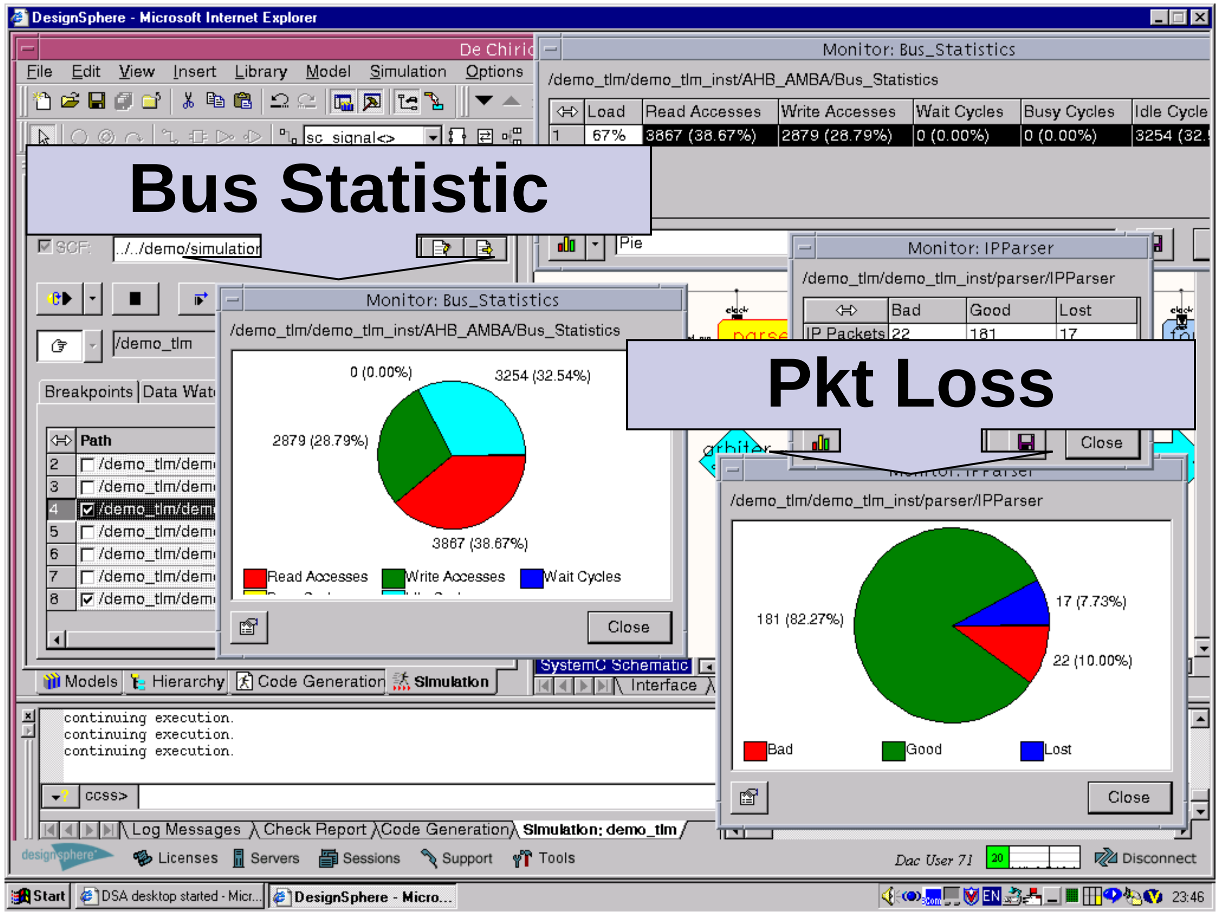
/demo_tlm/demo_tlm_inst/AHB_AMBA/Transfer_Trace

```
 102.5: /demo_tlm/demo_tlm_inst/sw read  addr: 0x0000102c;   /demo_tlm/demo_tlm_inst/mem data value: 0xffffffff  [ABM_OKAY]
 103.5: /demo_tlm/demo_tlm_inst/controller idle                 ;   /demo_tlm/demo_tlm_inst/mem data value: 0xffffffff  [ABM_OKAY]
 104.5: /demo_tlm/demo_tlm_inst/controller idle                 ;   /demo_tlm/demo_tlm_inst/AHB_AMBA/default_slave data value: no value
 105.5: /demo_tlm/demo_tlm_inst/sw read  addr: 0x00001028;   /demo_tlm/demo_tlm_inst/AHB_AMBA/default_slave data value: no value    [ABM_
 106.5: /demo_tlm/demo_tlm_inst/controller idle                 ;   /demo_tlm/demo_tlm_inst/mem data value: 0x00010203  [ABM_OKAY]
 107.5: /demo_tlm/demo_tlm_inst/controller idle                 ;   /demo_tlm/demo_tlm_inst/AHB_AMBA/default_slave data value: no value
 108.5: /demo_tlm/demo_tlm_inst/sw write addr: 0x00001028;   /demo_tlm/demo_tlm_inst/AHB_AMBA/default_slave data value: no value    [ABM_
 109.5: /demo_tlm/demo_tlm_inst/controller idle                 ;   /demo_tlm/demo_tlm_inst/mem data value: 0x10111213  [ABM_OKAY]
 110.5: /demo_tlm/demo_tlm_inst/controller idle                 ;   /demo_tlm/demo_tlm_inst/AHB_AMBA/default_slave data value: no value
 111.5: /demo_tlm/demo_tlm_inst/sw write addr: 0x0000102c;   /demo_tlm/demo_tlm_inst/AHB_AMBA/default_slave data value: no value    [ABM_
 112.5: /demo_tlm/demo_tlm_inst/local read  addr: 0x0000105c;   /demo_tlm/demo_tlm_inst/mem data value: 0x11111111  [ABM_OKAY]
```

Clear — Close

TL Modeling

System performance analysis


DesignSphere - Microsoft Internet Explorer
De Chiric
File Edit View Insert Library Model Simulation Options
Bus Statistic
Pkt Loss
Monitor: Bus_Statistics
Monitor: IPParser
Pie
Breakpoints Data Wat
Path
Models Hierarchy Code Generation Simulation
SystemC Schematic
Interface
Log Messages Check Report Code Generation Simulation: demo_tlm
Licenses Servers Sessions Support Tools
Dac User 71
Disconnect
Close
Start DSA desktop started - Micr... DesignSphere - Micro...
23:46

Monitor: Bus_Statistics

/demo_tlm/demo_tlm_inst/AHB_AMBA/Bus_Statistics

	Load	Read Accesses	Write Accesses	Wait Cycles	Busy Cycles	Idle Cycle
1	67%	3867 (38.67%)	2879 (28.79%)	0 (0.00%)	0 (0.00%)	3254 (32.

Pie chart: 0 (0.00%); 3254 (32.54%); 2879 (28.79%); 3867 (38.67%)

🟥 Read Accesses 🟩 Write Accesses 🟦 Wait Cycles

Monitor: IPParser

/demo_tlm/demo_tlm_inst/parser/IPParser

	Bad	Good	Lost
IP Packets	22	181	17

Pie chart: 181 (82.27%); 17 (7.73%); 22 (10.00%)

🟥 Bad 🟩 Good 🟦 Lost

```
continuing execution.
continuing execution.
continuing execution.
```

ccss>

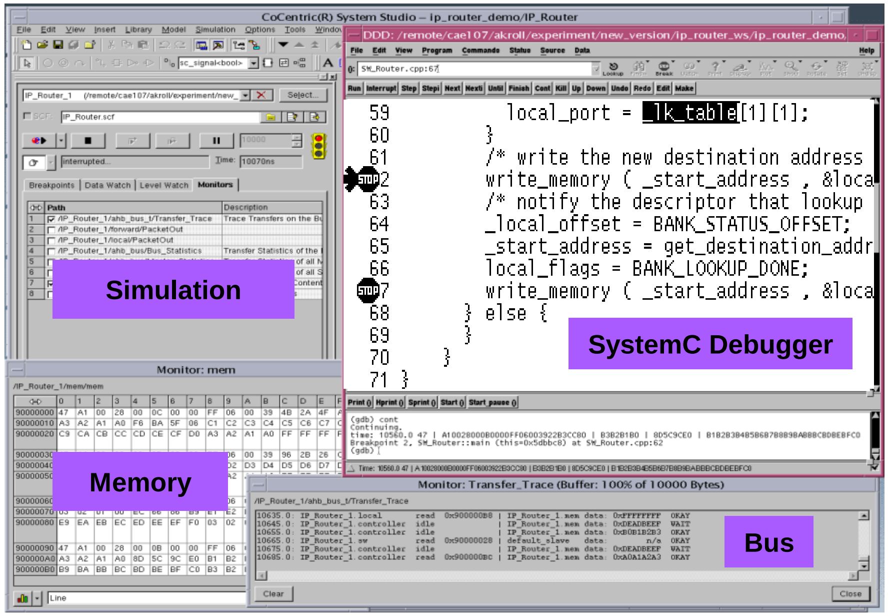
Early Software Verification

SW running on workstation with

- **annotated time or HW synchronized**

SW development

- **Algorithm**
- **Target indep. code**
- **Target code**



```
            local_port = _lk_table[1][1];
        }
        /* write the new destination address
        write_memory ( _start_address , &loca
        /* notify the descriptor that lookup
        _local_offset = BANK_STATUS_OFFSET;
        _start_address = get_destination_addr
        local_flags = BANK_LOOKUP_DONE;
        write_memory ( _start_address , &loca
      } else {
      }
    }
}
```

```
(gdb) cont
Continuing.
time: 10560.0 47 | A10028000B0000FF06003922B3CC80 | B3B2B1B0 | 8D5C9CE0 | B1B2B3B4B5B6B7B8B9BABBBCBDBEBFC0
Breakpoint 2, SW_Router::main (this=0x5dbbc8) at SW_Router.cpp:62
(gdb)
```

Monitor: Transfer_Trace (Buffer: 100% of 10000 Bytes)

/IP_Router_1/ahb_bus_t/Transfer_Trace

```
10635.0: IP_Router_1.local       read  0x900000B8 | IP_Router_1.mem data: 0xFFFFFFFF  OKAY
10645.0: IP_Router_1.controller  idle             | IP_Router_1.mem data: 0xDEADBEEF  WAIT
10655.0: IP_Router_1.controller  idle             | IP_Router_1.mem data: 0xB0B1B2B3  OKAY
10665.0: IP_Router_1.sw          read  0x90000028 | default_slave   data:        n/a  OKAY
10675.0: IP_Router_1.controller  idle             | IP_Router_1.mem data: 0xDEADBEEF  WAIT
10685.0: IP_Router_1.controller  read  0x900000BC | IP_Router_1.mem data: 0xA0A1A2A3  OKAY
```

Early SW Verification significantly shortens Integration and Validation

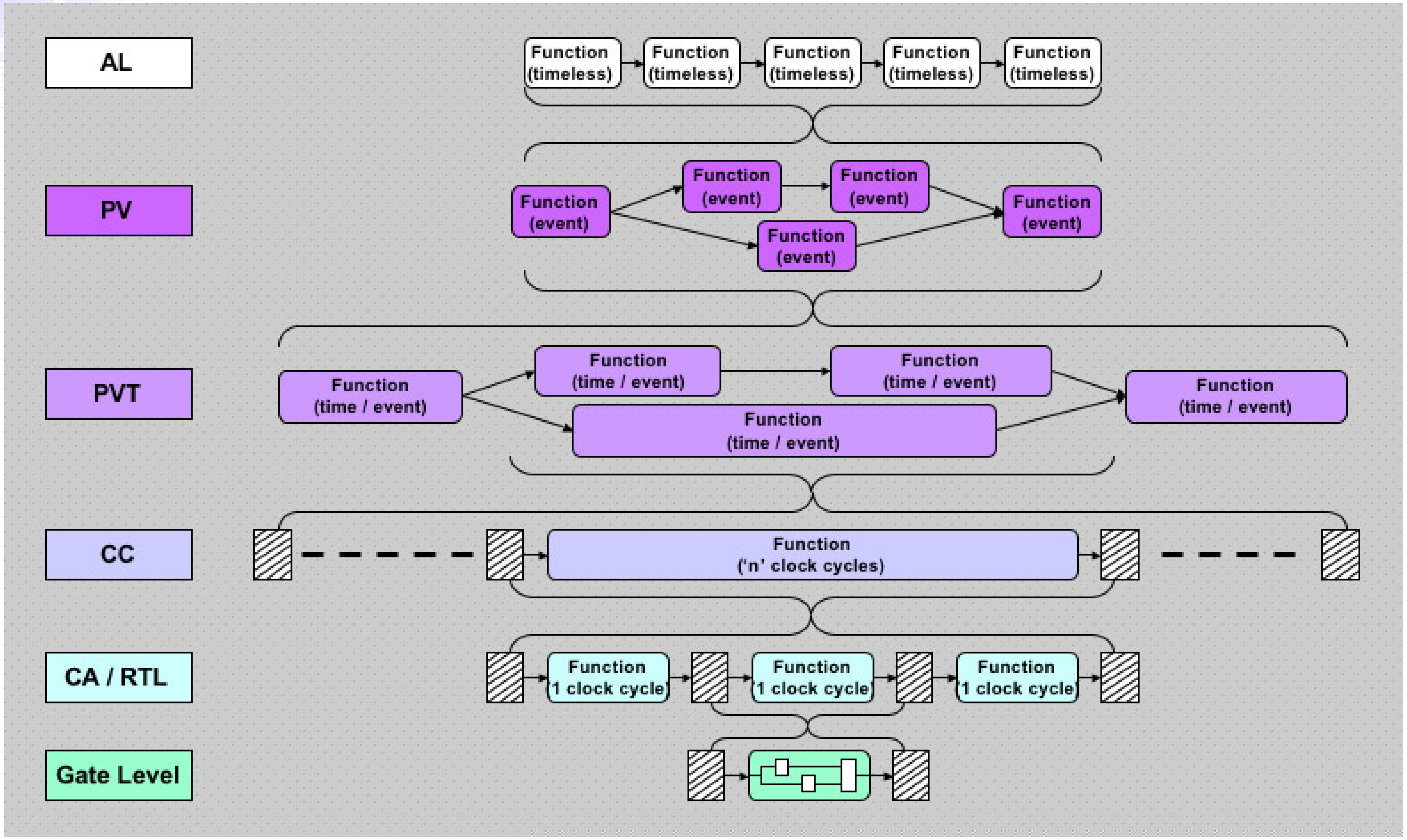
TL Modeling - Benefits

- High Performance
  - potential for 100x speed over RTL
- Early architectural closure
  - A platform for software developers to write code
  - Model for early system analysis
- Reuse of functional test bench
- Architecture Verification
  - Analysis of cache/memory architecture
  - System latency
- TL model library

More Details about More General Transaction-Level Modeling

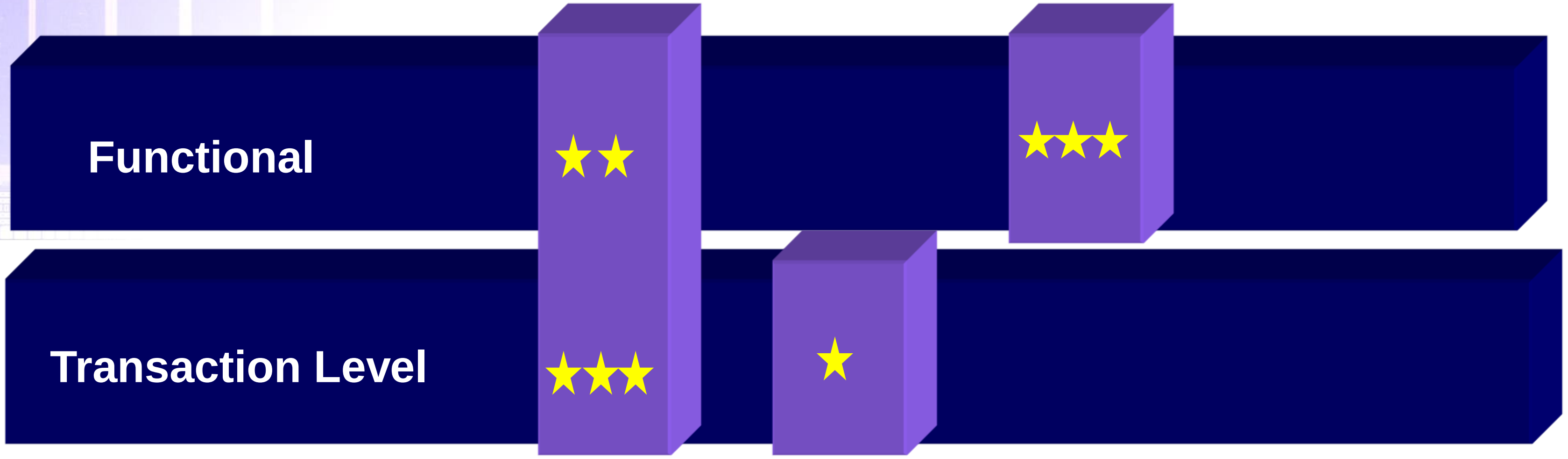
- Abstractions
  - Algorithmic level (AL)
    - Architecture/implementation independent
  - Programmers View (PV)
    - Bit-true representation of the HW, register accurate, no detailed timing
  - Programmer View + Timing (PVT)
    - Same as PV plus detailed timing and synchronization (cycle approximate in most cases)
  - Cycle Accurate (CA)
    - Clocked abstraction, interfaces and transactions
  - RTL
    - Clocked abstraction, actual chip signals

Transaction-Level Modeling Abstractions



Languages

★★★ excellent
★★ good
★ ok



SystemC

System Verilog

Ptolemy
Matlab

Register Transfer Level Modeling

Functionality	Yes
Cycle Accuracy	Yes
Timing	Yes
Pin Accuracy	Yes
Communication	Shared
Channel	Signals only
Parameters	Yes

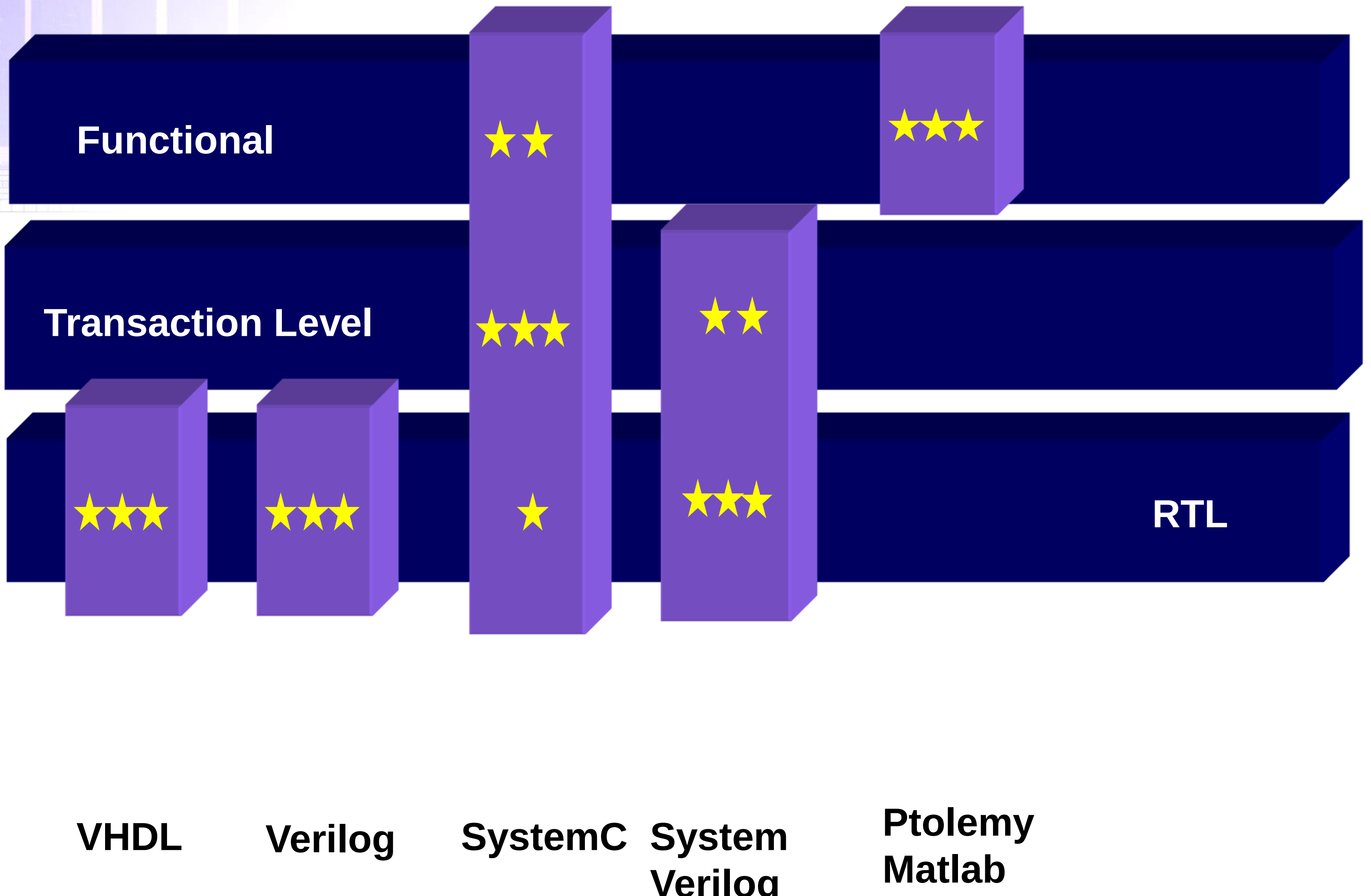
RT Level Models

- Functionality of the device
- All signal interactions with the bus
  - Databus
  - Address bus
  - Control
    - signal characteristics (active high/low)
    - reset characteristics
    - bus responses
    - arbitration protocols
- Accurate timing information of signals

RTL - Benefits

- Well understood semantics
- Popular languages
  - Verilog, VHDL
  - SystemC (not primarily targeted at RTL)
- Synthesize-ability
  - well defined synthesis tools and methodology
- Analysis capabilities
  - accurate timing analysis and verification tools
- RTL models can be plugged into TL models with adaptors

Languages



★★★ excellent
★★ good
★ ok
Functional
Transaction Level
RTL
★★
★★★
★★★
★★
★★★
★★★
★
★★★
VHDL
Verilog
SystemC
System Verilog
Ptolemy Matlab

System-C or SystemVerilog?

- System-C
  - A library
  - Built on C++
  - Can be support by your C compiler
  - For advanced tools, will be supported by Cadence, Coware, MentorGrahics, Synopsys, …
- SystemVerilog
  - A new language
  - Next generation of Verilog
  - Supported by your Verilog simulator and synthesizer

SystemC
http://www.systemc.org

- 1997: Scenic Design Framework (Synopsys, UC Irvine, DAC'97)
- 1999: Open SystemC Initiative (Synopsys, CoWare)
- 1999: SystemC v0.9, C++ class library
- 2000: SystemC 1.0
- 2000: Cadence joins OSCI
- 2001: Mentor Graphics joins OSCI
- 2001: SystemC Release 2.0
  - higher levels of abstraction
  - interfaces, channels, ports
  - stepwise refinement
- 2002: SystemC Release 2.0.1

SystemC

http://www.systemc.org ➔ http://www.accellera.org/

- 2003: SystemC Verification Library
  - Cadence TestBuilder
  - Constrained randomization
  - Weighted randomization
  - Introspection
  - begin of standard for HDL integration
- 2005: SystemC 2.1 and TLM 1.0
- 2005: IEEE approves the IEEE 1666™ -2005 standard for SystemC
- 2007: SystemC 2.2
- 2008: TLM 2.0
- 2009: TLM 2.0.1
- 2010: AMS 1.0
- 2011: IEEE approves the IEEE 1666–2011 standard for System
- 2011: Accellera and Open SystemC Initiative (OSCI) approve merger, unite to form Accellera Systems Initiative
- 2012: SystemC 2.3
- 2014: SystemC 2.3.1

SystemVerilog (1)
http://www.systemverilog.org

- 1984: Gateway Design Automation introduced Verilog
- 1989: Gateway merged into Cadence Design Systems
- 1990: Cadence put Verilog HDL into the public domain
- 1993: OVI enhanced the Verilog language - not well accepted
- 1995: IEEE standardized the Verilog HDL (IEEE 1364-1995)
- 2001: IEEE standardized the Verilog IEEE Std1364-2001
- 2002: IEEE standardized the Verilog IEEE Std1364.1-2002
- 2002: Accellera standardized SystemVerilog 3.0
  - Accellera is the merged replacement of OVI & VHDL International (VI)
- 2003: Accellera standardized SystemVerilog 3.1
- 2005: IEEE approves the IEEE 1800™ -2005 Unified Hardware Design, Specification and Verification Language."

SystemVerilog (2)

- SystemVerilog is *revolutionary evolution* of Verilog

- Verilog 1.0 - IEEE 1364-1995 "Verilog-1995" standard
  - The first IEEE Verilog standard
- Verilog 2.0 - IEEE 1364-2001 "Verilog-2001" standard
  - The second generation IEEE Verilog standard
  - Significant enhancements over Verilog-1995
- SystemVerilog 3.x - Accellera extensions to Verilog-2001
  - A third generation Verilog standard
  - DAC-2002 - SystemVerilog 3.0
  - DAC-2003 - SystemVerilog 3.1
  - DAC-2004 - SystemVerilog 3.1a offered to IEEE P1800

SystemVerilog (3)

SystemVerilog

assertions
test program blocks
clocking domains
process control

mailboxes
semaphores
constrained random values
direct C function calls

interfaces
nested hierarchy
unrestricted ports
automatic port connect
enhanced literals
time values and units
specialized procedures

dynamic processes
2- state modeling
packed arrays
array assignments
enhanced event control
unique/ priority case/ if
root name space access

classes
inheritance
strings

dynamic arrays
associative arrays
references

from C/C++

int
shortint
longint
byte
shortreal
void
alias

globals
enum
typedef
structures
unions
casting
const

break
continue
return
do? while
++ -- += -= *= /=
>>= <<= >>>= <<<=
&= |= ^= %=

Verilog 2001

ANSI C style ports
generate
localparam
constant functions

standard file I/ O
$value$ plusargs
`ifndef `elsif `line
@*

(* attributes *)
configurations
memory part selects
variable part select

multi dimensional arrays
signed types
automatic
** (power operator)

Verilog 1995

modules
parameters
function/tasks
always @
assign

$finish $fopen $fclose
$display $write
$monitor
`define `ifdef `else
`include `timescale

initial
disable
events
wait # @
fork? join

wire reg
integer real
time
packed arrays
2D memory

begin end
while
for forever
if else
repeat

+ = * /
%
>> <<

SystemC

- Not a new language
- A special class library
- Based on C++
  - Includes all the advantages/disadvantages of C++
- Good reference implementation
- C++ compatibility supports SW compatibility
- Only limited path to implementation
- TLM methodology and experience exists
- Oriented towards HDS verification, architecture exploration, and fast higher level simulation

SystemVerilog

- System level extension of Verilog towards system and transaction level modeling
- Relevant semantics part of the language
- Clean and concise
- Excellent LRM (language reference manual)
- Elaboration and compiler can do multiple checks
- Verilog compatibility guarantees legacy compatibility and full path to implementation
- No programming language
- Co-existence with C++

A General Thinking of SystemC and SystemVerilog

	SystemC	SystemVerilog
Architectural Design	👍👍👍	
Architectural Verification & HW/SW Co-Verification	👍👍👍	👍
RTL-to-Gates Design		👍👍👍
RTL-to-Gates Verification	👍	👍👍👍

Outline

- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
  - IBM SEAS
  - Synopsys's solution
  - ARM's solution
  - High level synthesis tools
- Conclusion

ESL Toolset Should Consist of

- Formal system requirement capture, analysis, and traceability tools
- Architectural modeling, analysis, optimization, and verification environment
- Simulators and abstract processor models for software validation
- High-level synthesis and configurable IP approaches to fixed-function hardware development
- Architectural development, synthesis, and configurable IP design approaches to programmable hardware development
- Diverse design aids such as system-level modeling libraries and model generation tools

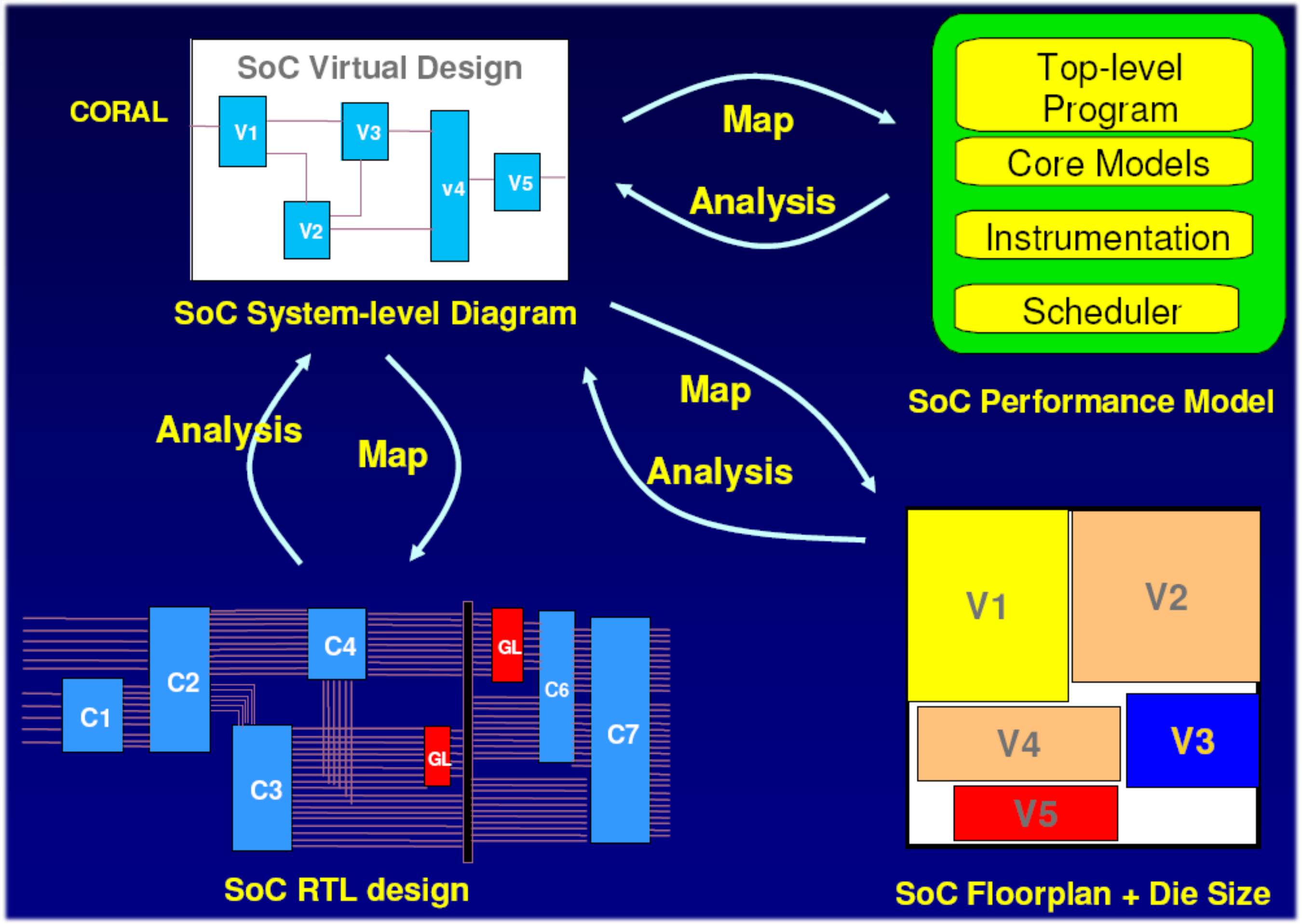
ESL Flow Supported by the Tools

- Specifications and modeling
- Pre-partitioning analysis
- Partitioning
  - Hardware/software partition
  - Hardware partition
  - Software partition
- Post-partition analysis and debug
- Post-partition verification
- Hardware implementation
- Software implementation
- Hardware/software co-verification

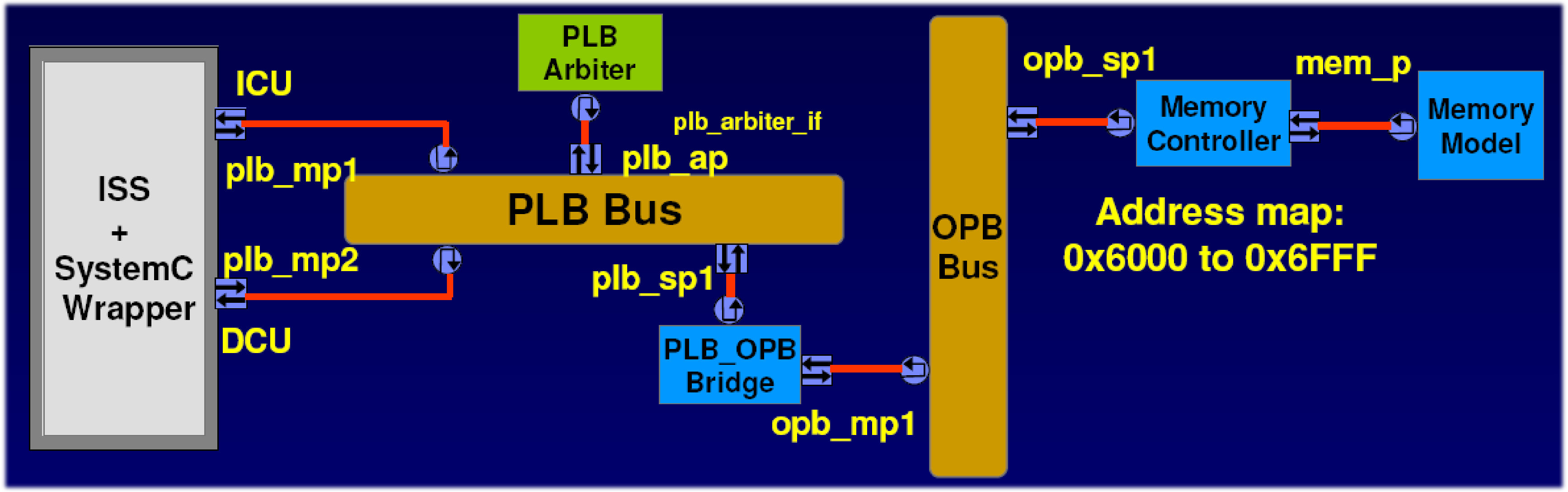
Some Existing System-Level Design Tools

- IBM SEAS
- Synopsys's solution
- ARM's solution
- High level synthesis tools

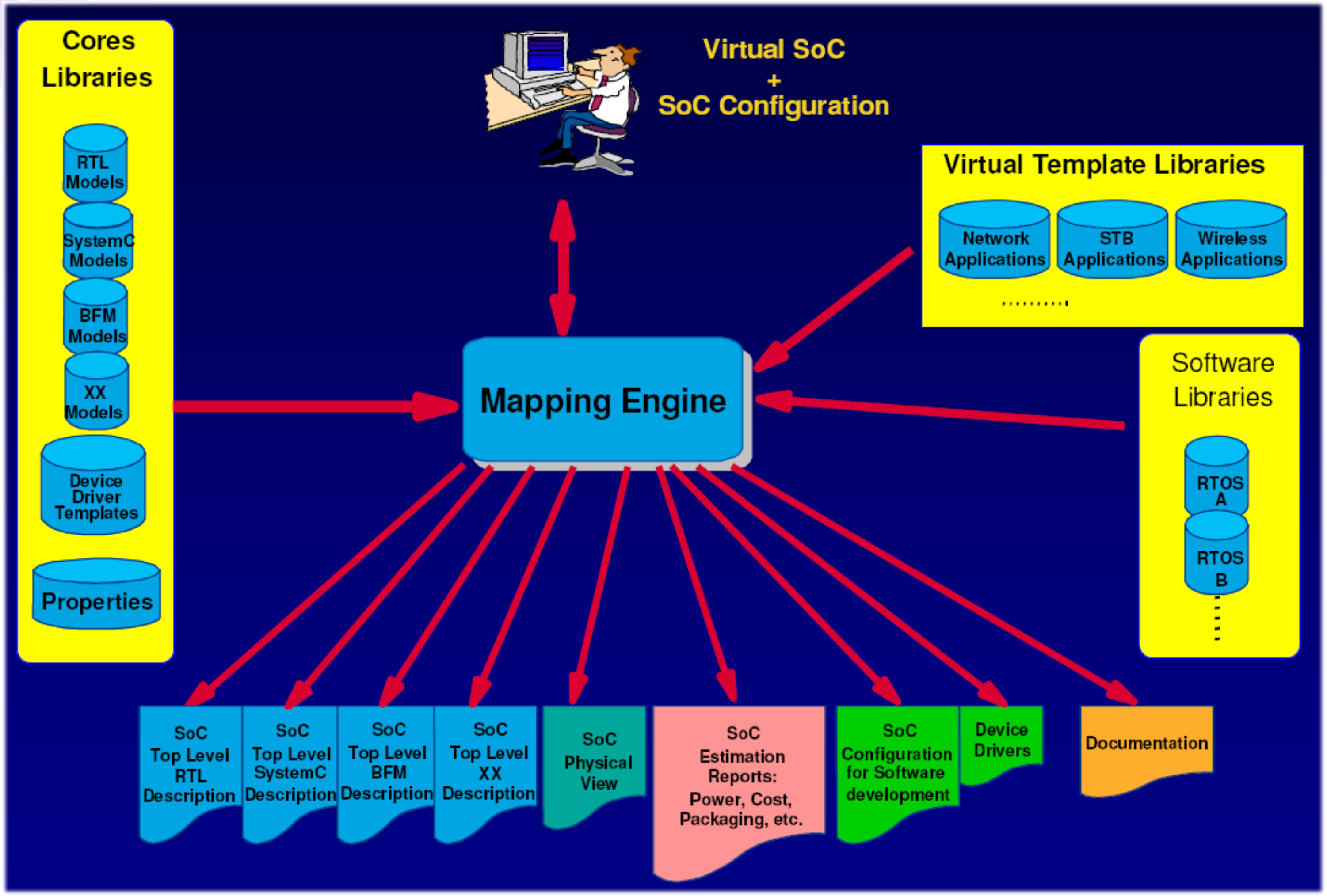
IBM SEAS: a System for Early Analysis of SoCs



SEAS Architecture (1)

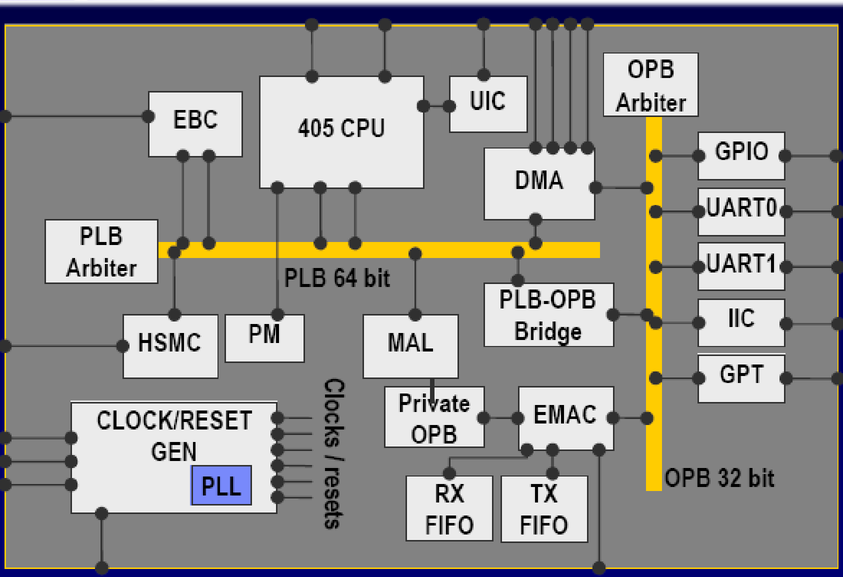


SEAS Architecture (2)

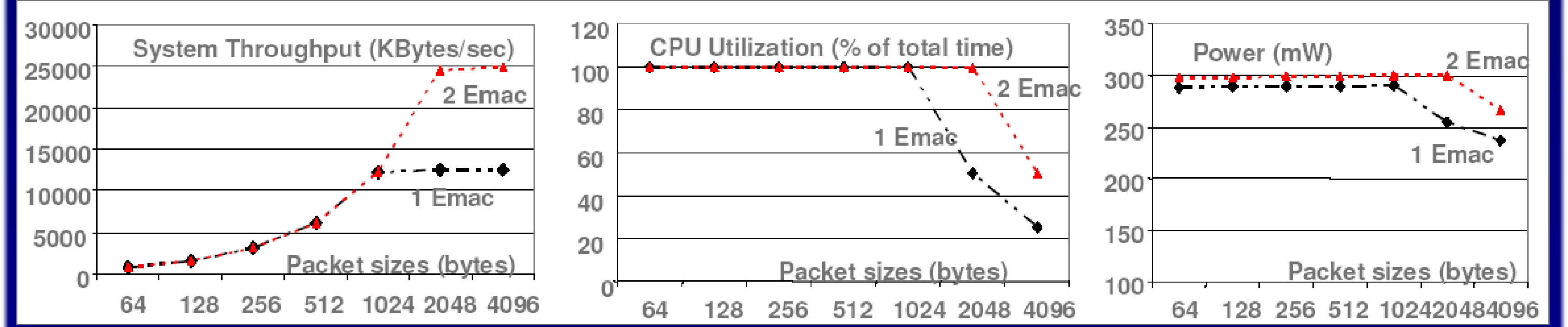


Cores Libraries
RTL Models
SystemC Models
BFM Models
XX Models
Device Driver Templates
Properties
Virtual SoC + SoC Configuration
Virtual Template Libraries
Network Applications
STB Applications
Wireless Applications
....
Software Libraries
RTOS A
RTOS B
Mapping Engine
SoC Top Level RTL Description
SoC Top Level SystemC Description
SoC Top Level BFM Description
SoC Top Level XX Description
SoC Physical View
SoC Estimation Reports: Power, Cost, Packaging, etc.
SoC Configuration for Software development
Device Drivers
Documentation

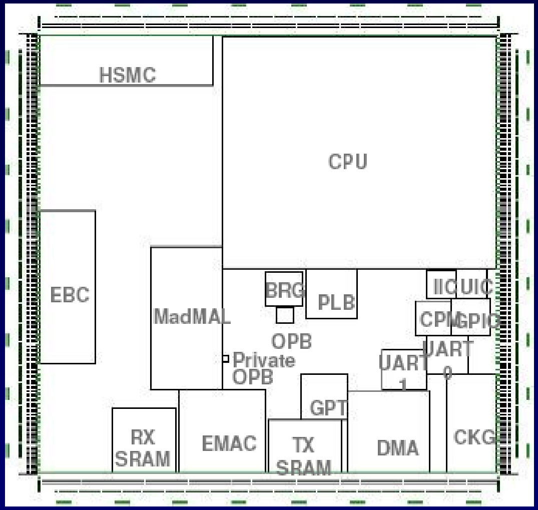
SEAS Experiment



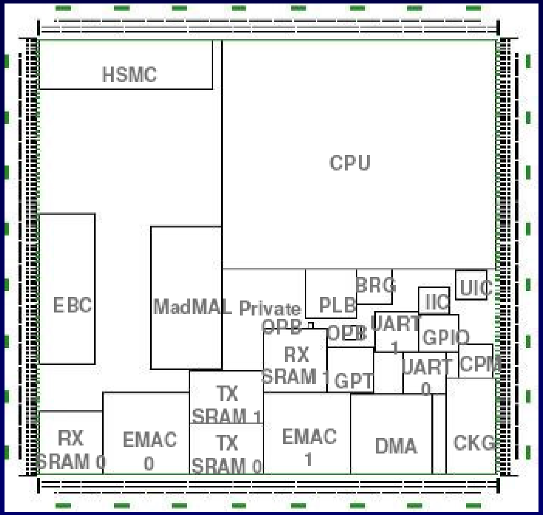
- 405PBD
  - Ethernet Subsystem
    - 1 EMAC
    - 1 Madmal
- Change to improve performance
  - Added an extra EMAC + Fifos
- Measure effects on die-size, fp, timing, power



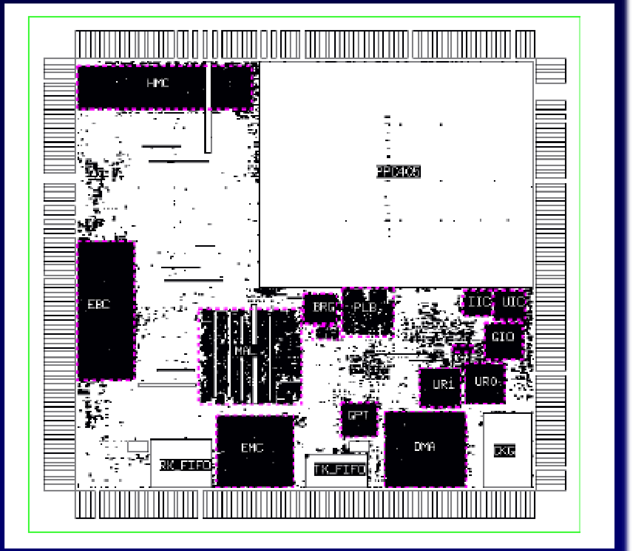
SEAS Experiment



1 EMAC
6.05mmx6.05mm



2 EMACs
6.05mmx6.05mm



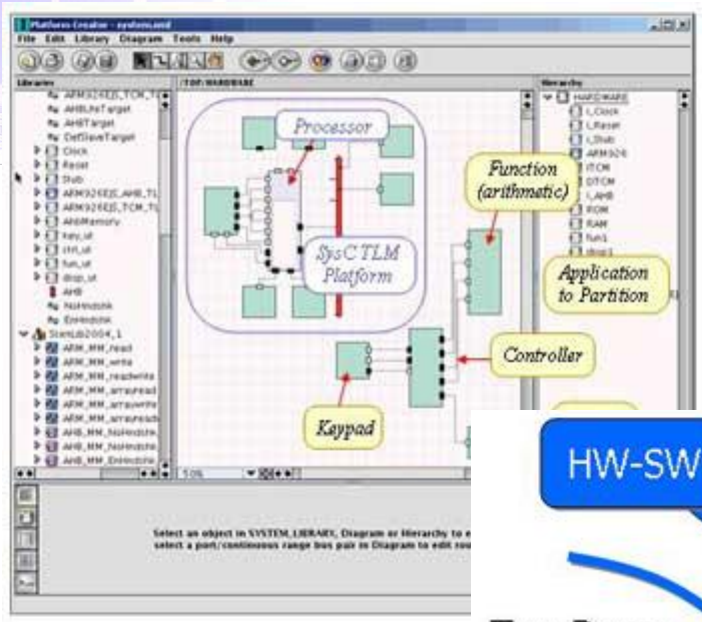
Real Design Layout (1 EMAC)
6.05mmx6.05mm

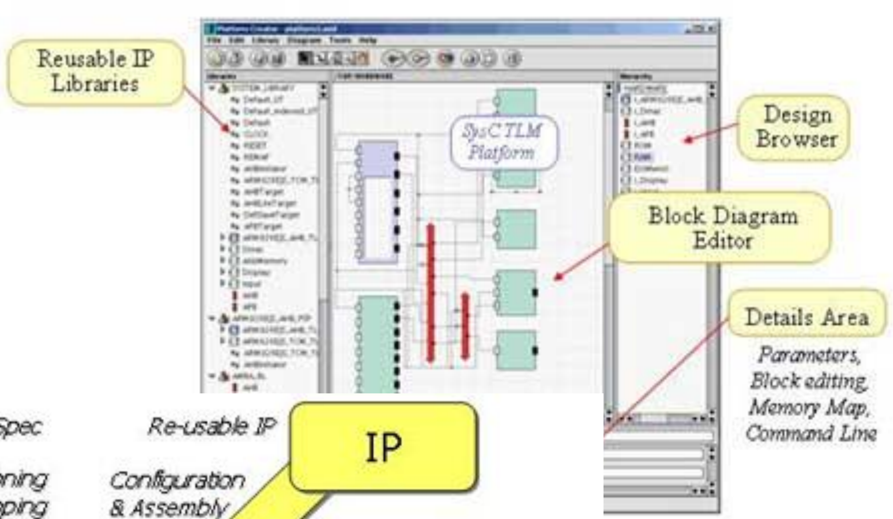
- **Results**
  - **Two Emac solution delivered the required performance**
  - **Could fit in the same die-size as the original one**
  - **Met the same timing requirements as the original one**

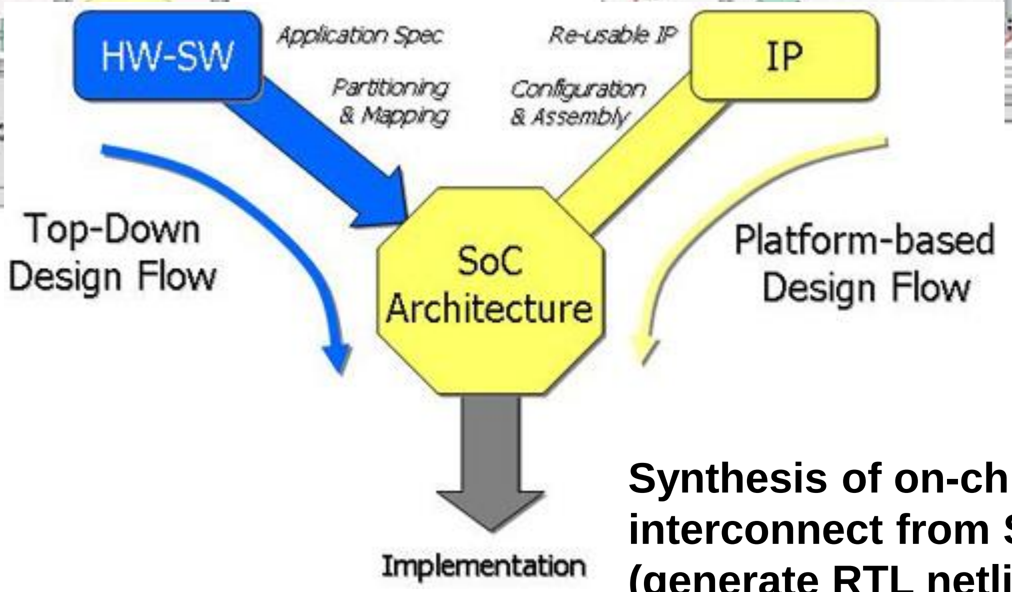
Synopsys's Solutions

- High-level block design
  - System Studio, SPW, Synphony C compiler, Processor Designer, …
- Architecture design
  - Platform Architect
- Virtual platform
  - Innovator, Platform Architect
- FPGA-based prototyping
  - HAPS, Certify, Synplify Premier, Identify

Platform Creator



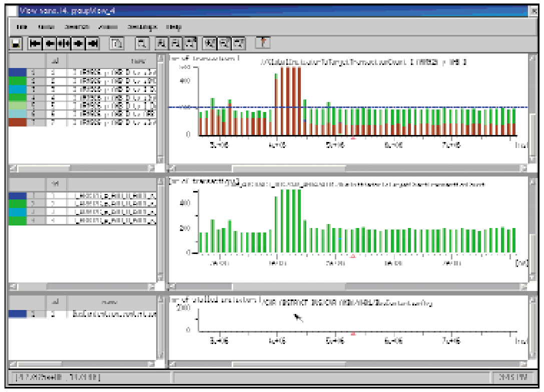


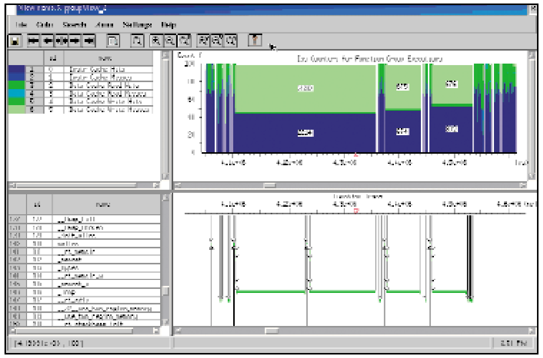


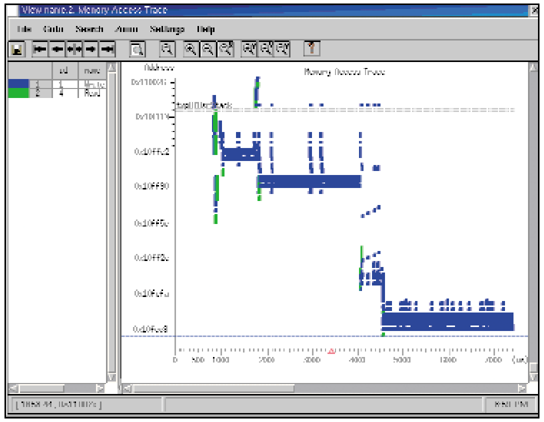
•HW/SW partition
•HW/SW co-design

Synthesis of on-chip interconnect from SystemC (generate RTL netlist)

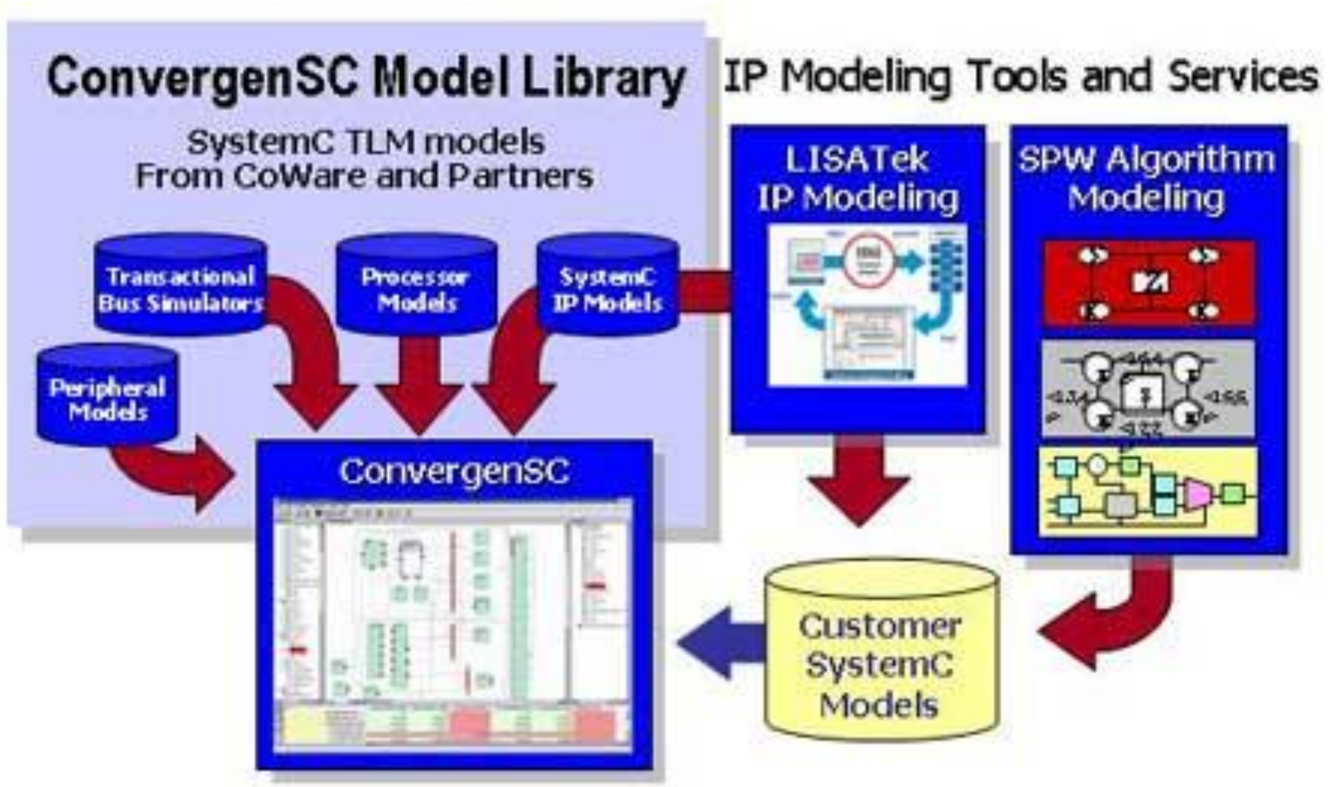
System-Level Analysis


Transaction Counts and Bus Contention — "Which masters and slaves should be on which bus layer?"

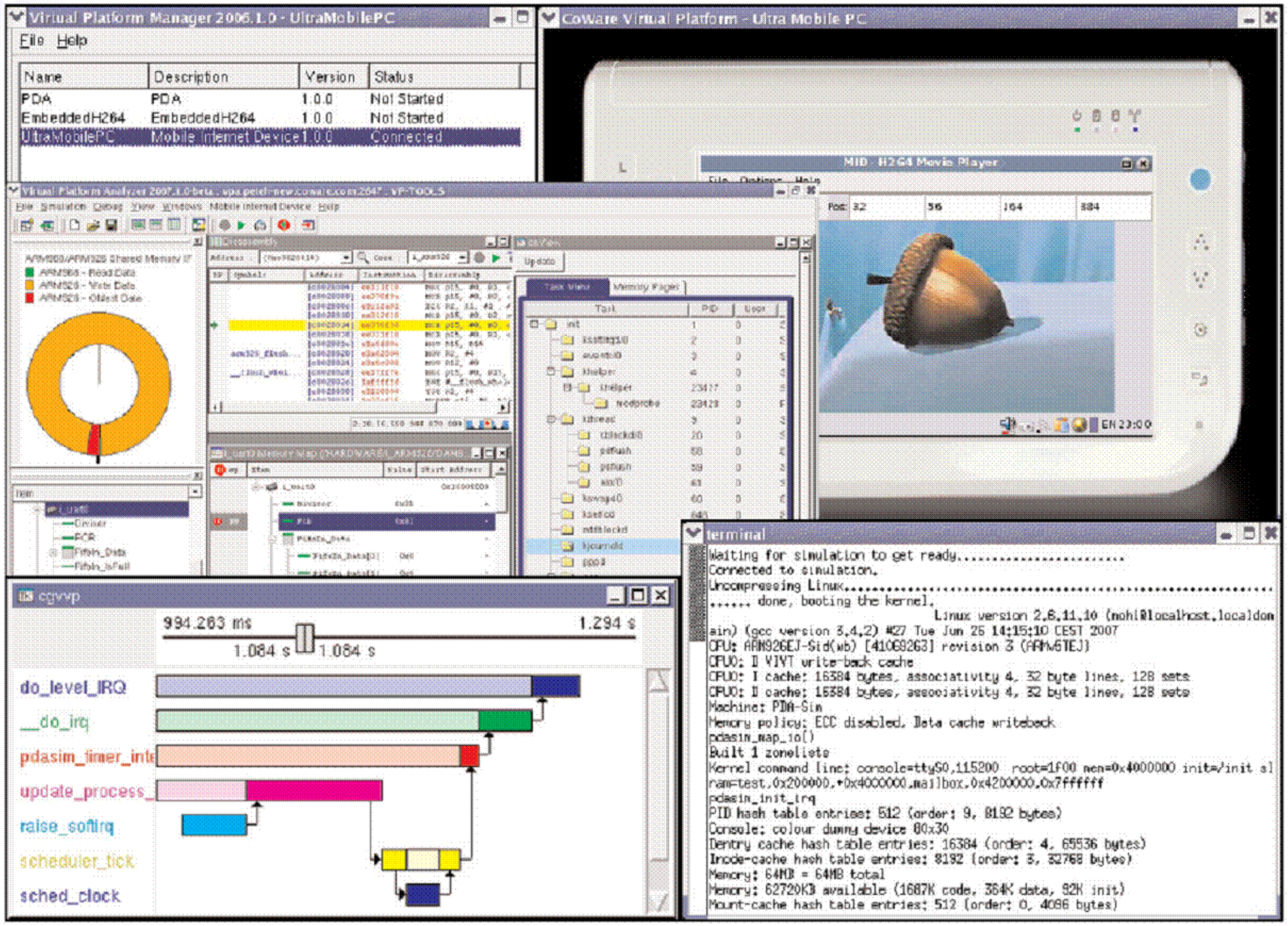

Cache Hits/Misses and SW Task Gantt — "Is the cache size correct?"


Memory Reads and Writes — "Is the memory architecture optimal?"

CoWare Model Library



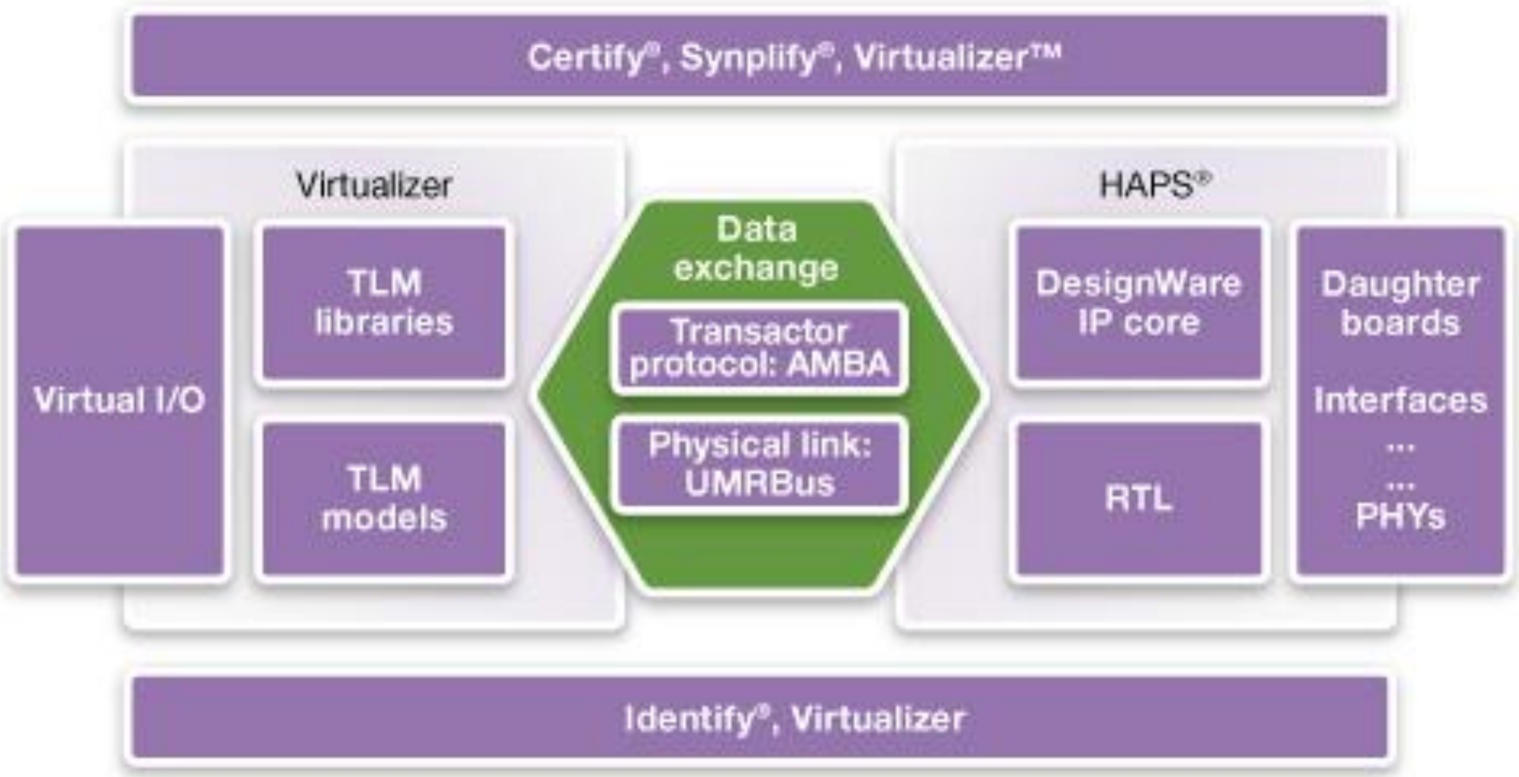
CoWare Virtual Platform



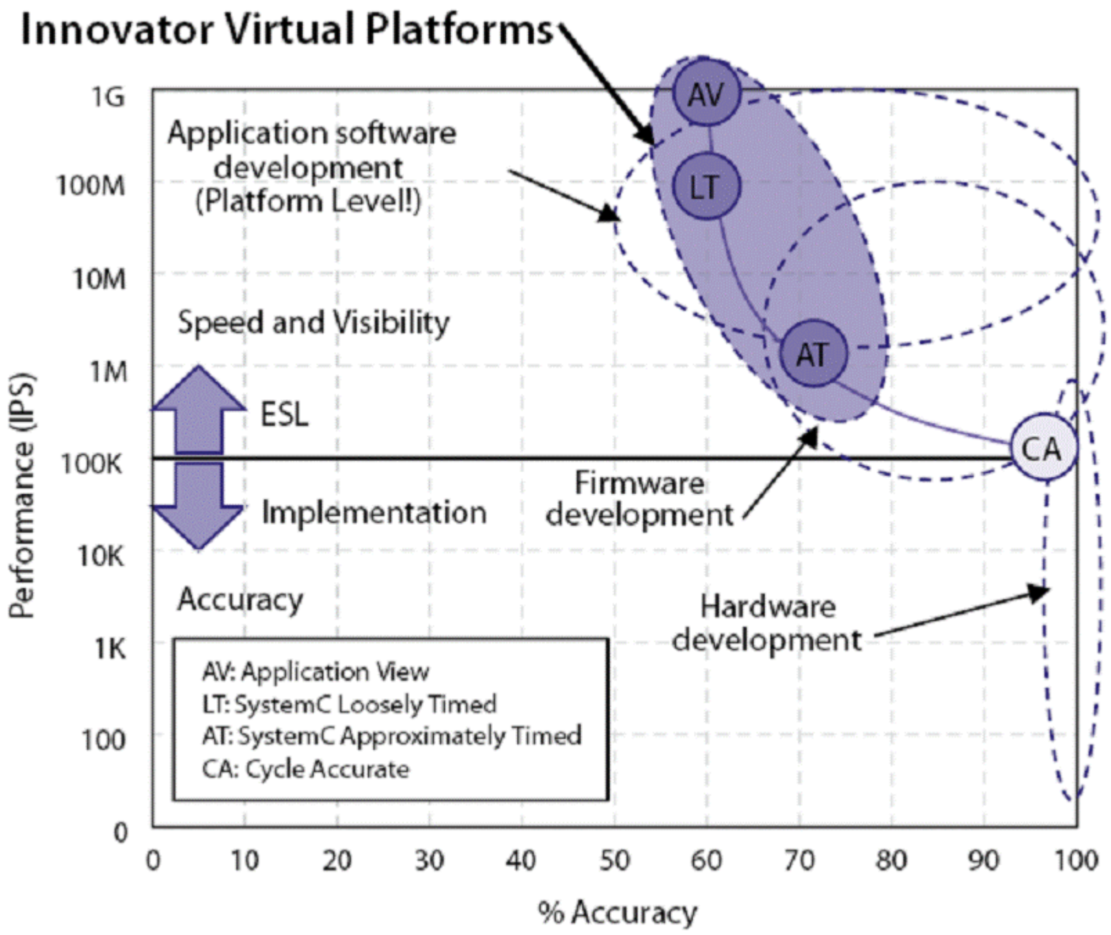
Name	Description	Version	Status
PDA	PDA	1.0.0	Not Started
EmbeddedH264	EmbeddedH264	1.0.0	Not Started
UltraMobilePC	Mobile Internet Device	1.0.0	Connected

```
Waiting for simulation to get ready.........................
Connected to simulation.
Uncompressing Linux.................................................................
...... done, booting the kernel.
                                             Linux version 2.6.11.10 (nohl@localhost.localdom
ain) (gcc version 3.4.2) #27 Tue Jun 26 14:15:10 CEST 2007
CPU: ARM926EJ-Sid(wb) [41069263] revision 3 (ARMv5TEJ)
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
CPU0: D cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
Machine: PDA-Sim
Memory policy: ECC disabled, Data cache writeback
pdasim_map_io()
Built 1 zonelists
Kernel command line: console=ttyS0,115200  root=1f00 mem=0x4000000 init=/init sl
ram=test,0x200000,+0x4000000,mailbox,0x4200000,0x7ffffff
pdasim_init_irq
PID hash table entries: 512 (order: 9, 8192 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 64MB = 64MB total
Memory: 62720KB available (1687K code, 364K data, 92K init)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
```

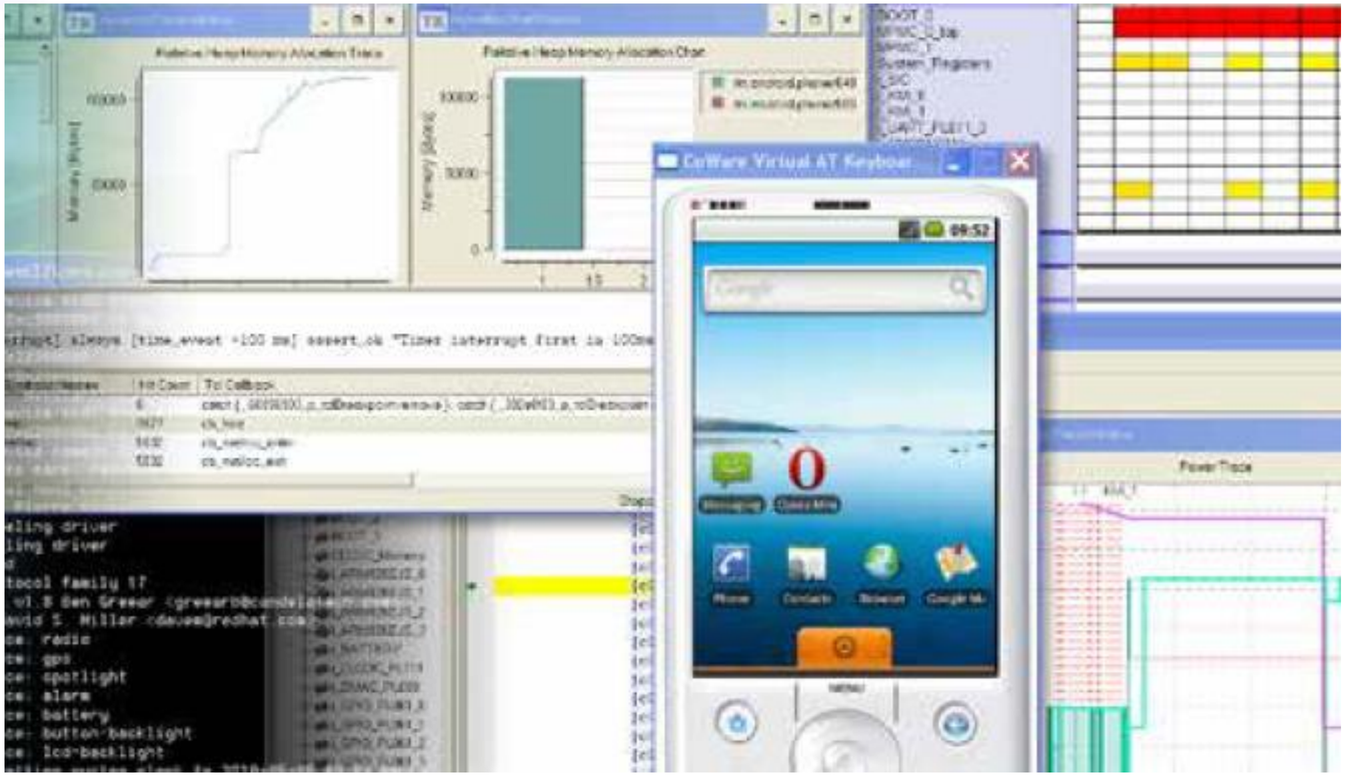
Synopsys's Solution



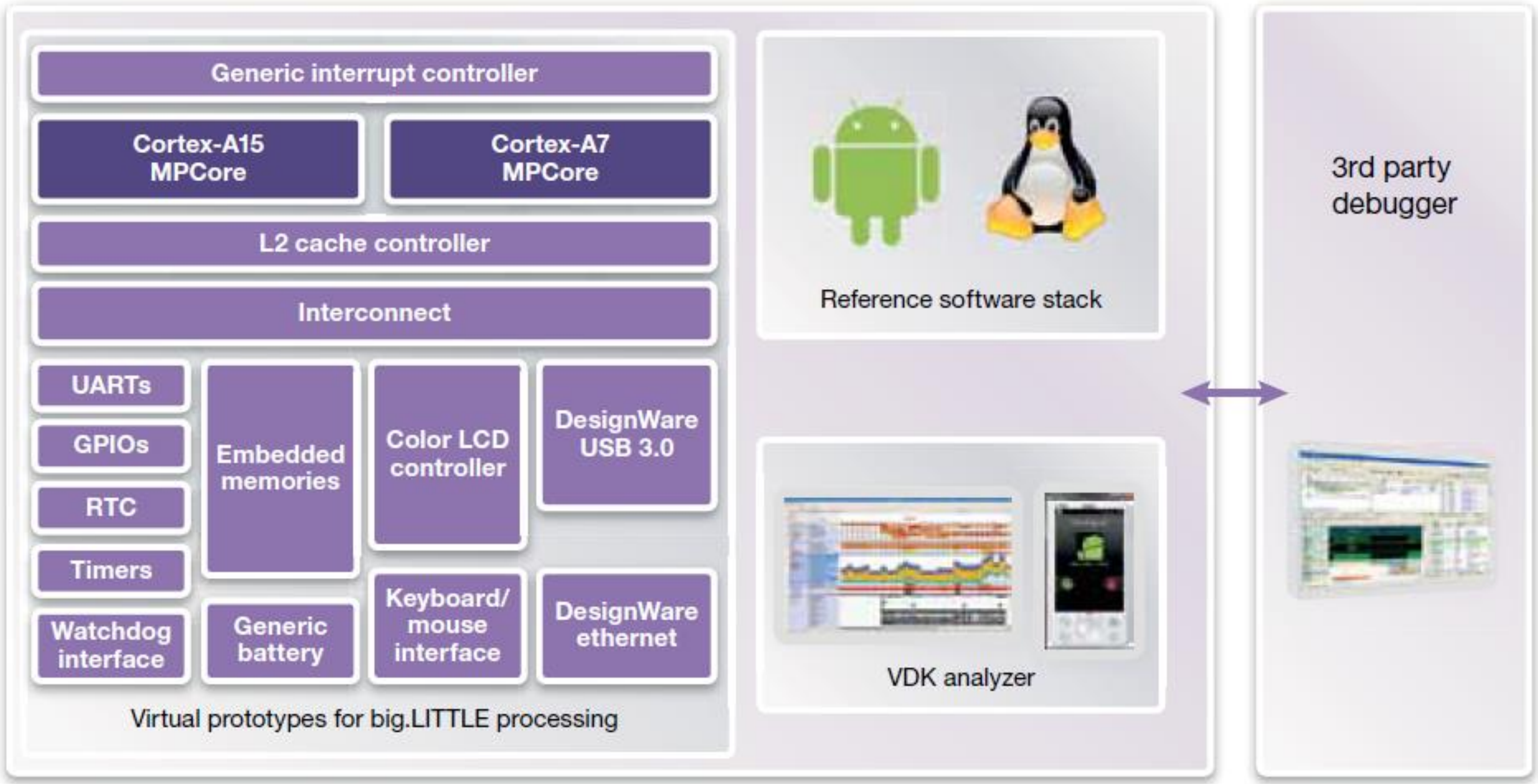
Synopsys Virtualizer



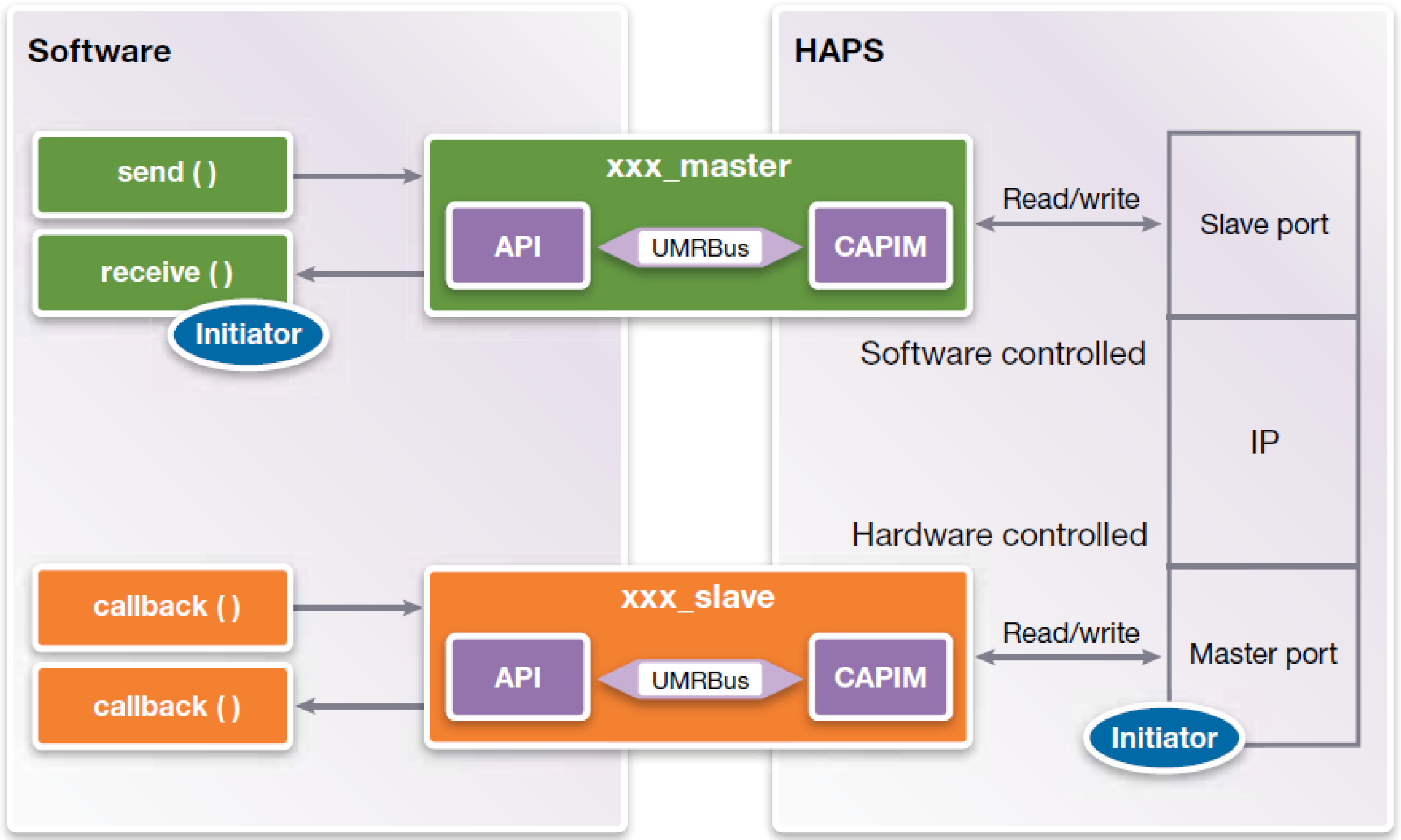
Synopsys Virtualizer



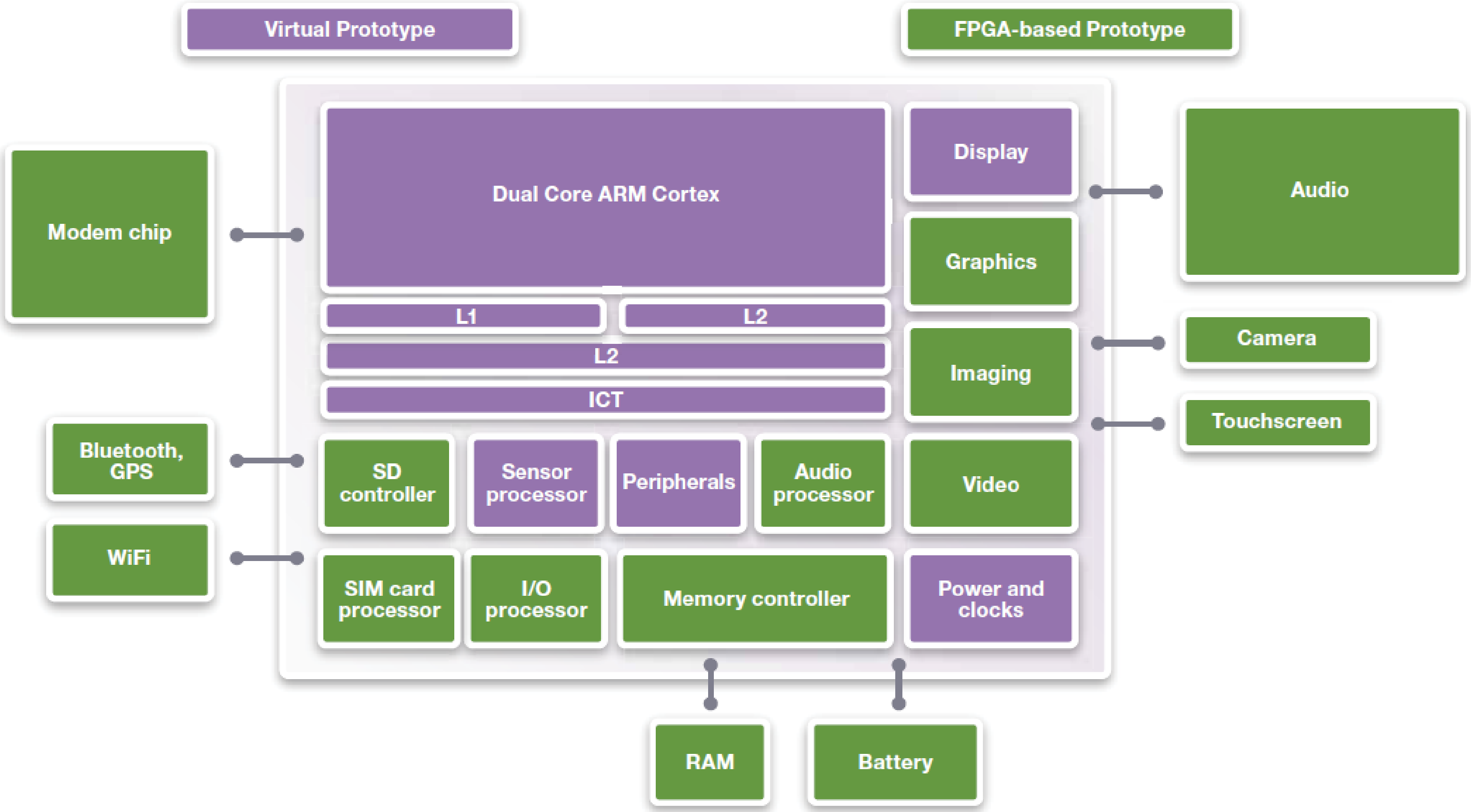
Example VDK for ARMv7



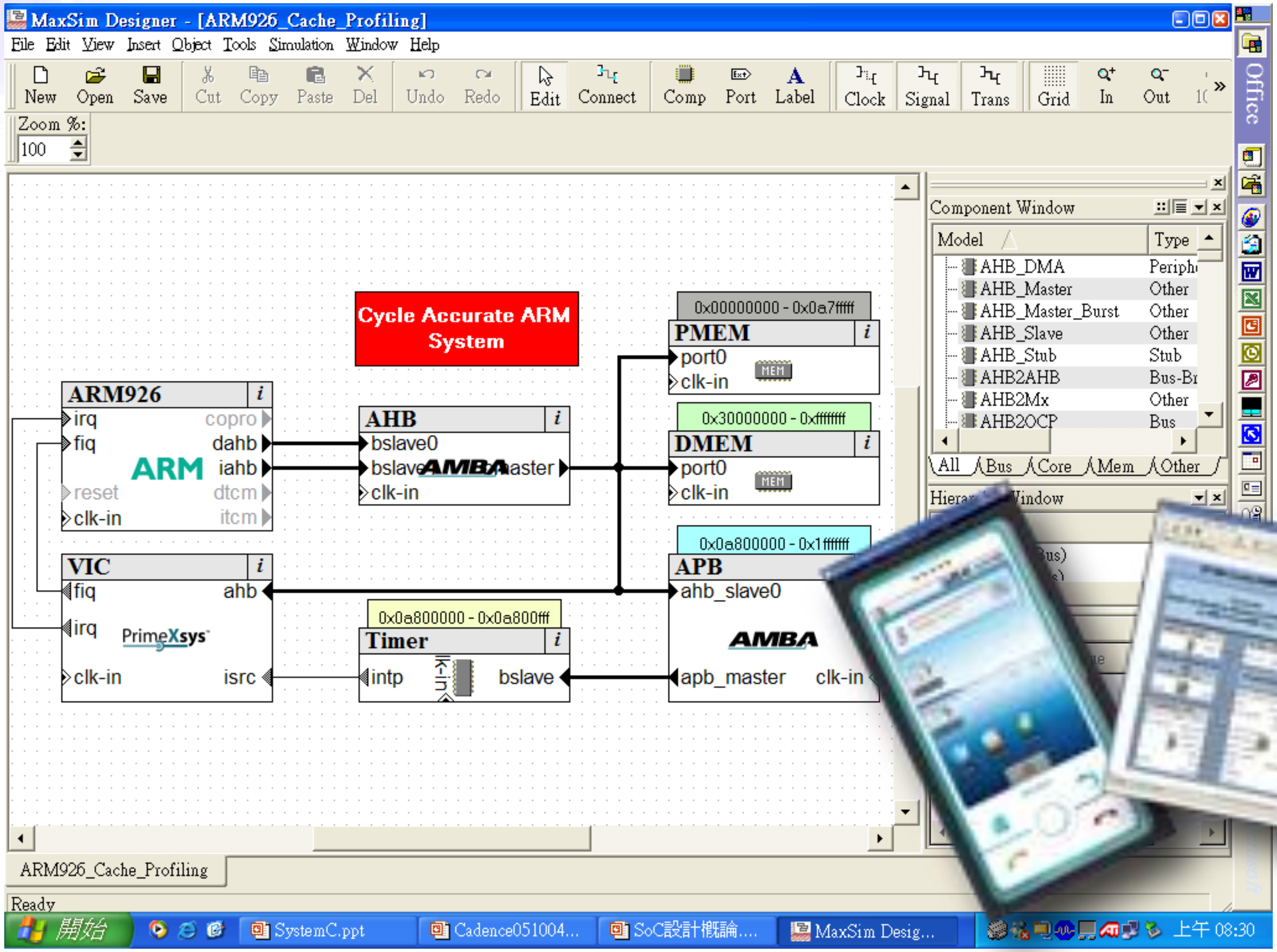
Synopsys Hybrid Prototype System



Synopsys Hybrid Prototype System



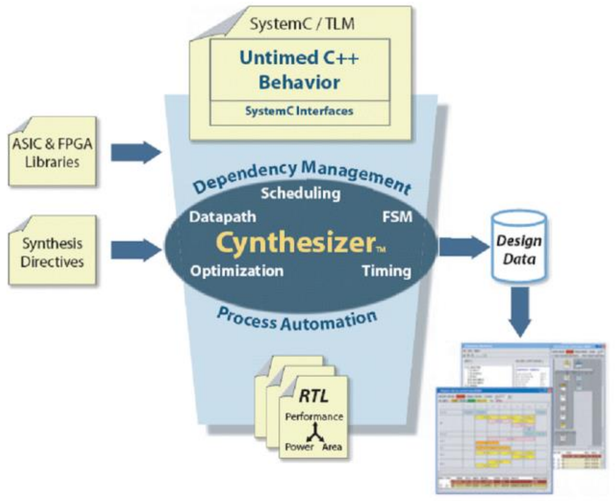
ARM Fast Models
Carbon SoC Designer

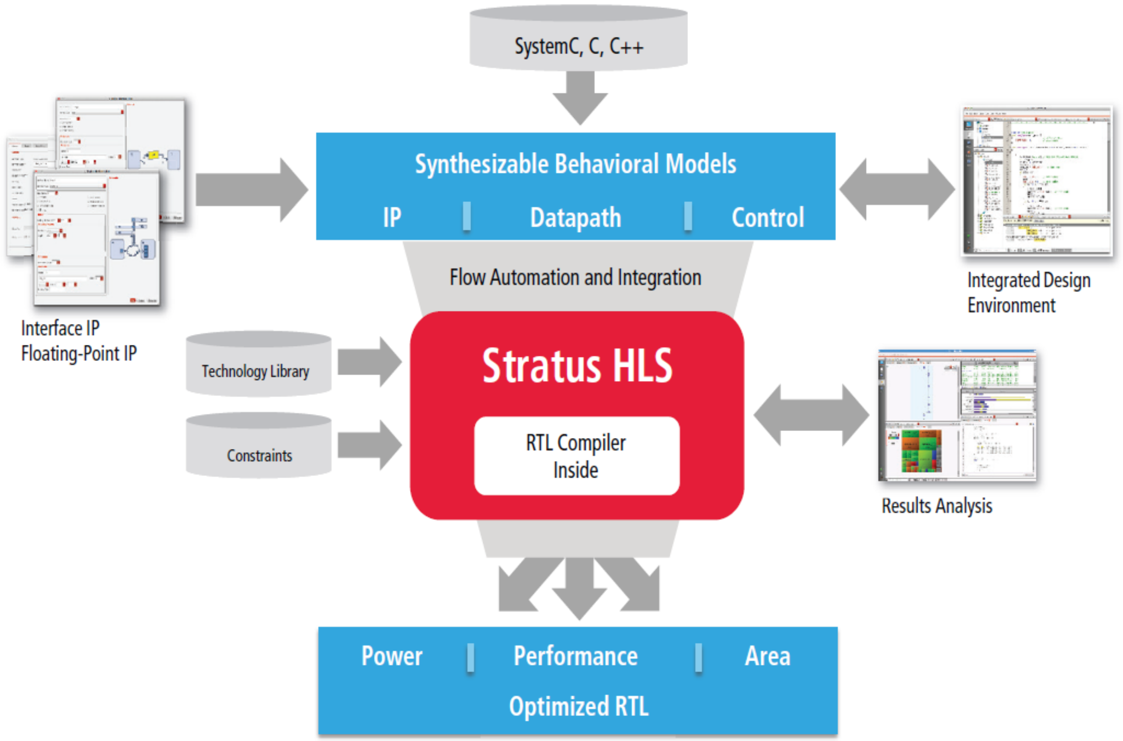

MaxSim Designer - [ARM926_Cache_Profiling]
File Edit View Insert Object Tools Simulation Window Help
New Open Save Cut Copy Paste Del Undo Redo Edit Connect Comp Port Label Clock Signal Trans Grid In Out
Zoom %: 100
Cycle Accurate ARM System
ARM926 irq fiq reset clk-in copro dahb iahb dtcm itcm
VIC fiq irq clk-in ahb isrc
AHB bslave0 bslave master clk-in
Timer intp bslave
0x0a800000 - 0x0a800fff
PMEM port0 clk-in
0x00000000 - 0x0a7fffff
DMEM port0 clk-in
0x30000000 - 0xffffffff
APB ahb_slave0 apb_master clk-in
0x0a800000 - 0x1ffffff
Component Window
Model Type
AHB_DMA Periph
AHB_Master Other
AHB_Master_Burst Other
AHB_Slave Other
AHB_Stub Stub
AHB2AHB Bus-Br
AHB2Mx Other
AHB2OCP Bus
All Bus Core Mem Other
Hierarchy Window
ARM926_Cache_Profiling
Ready
開始 SystemC.ppt Cadence051004... SoC設計概論.... MaxSim Desig... 上午 08:30

High Level Synthesis Tools

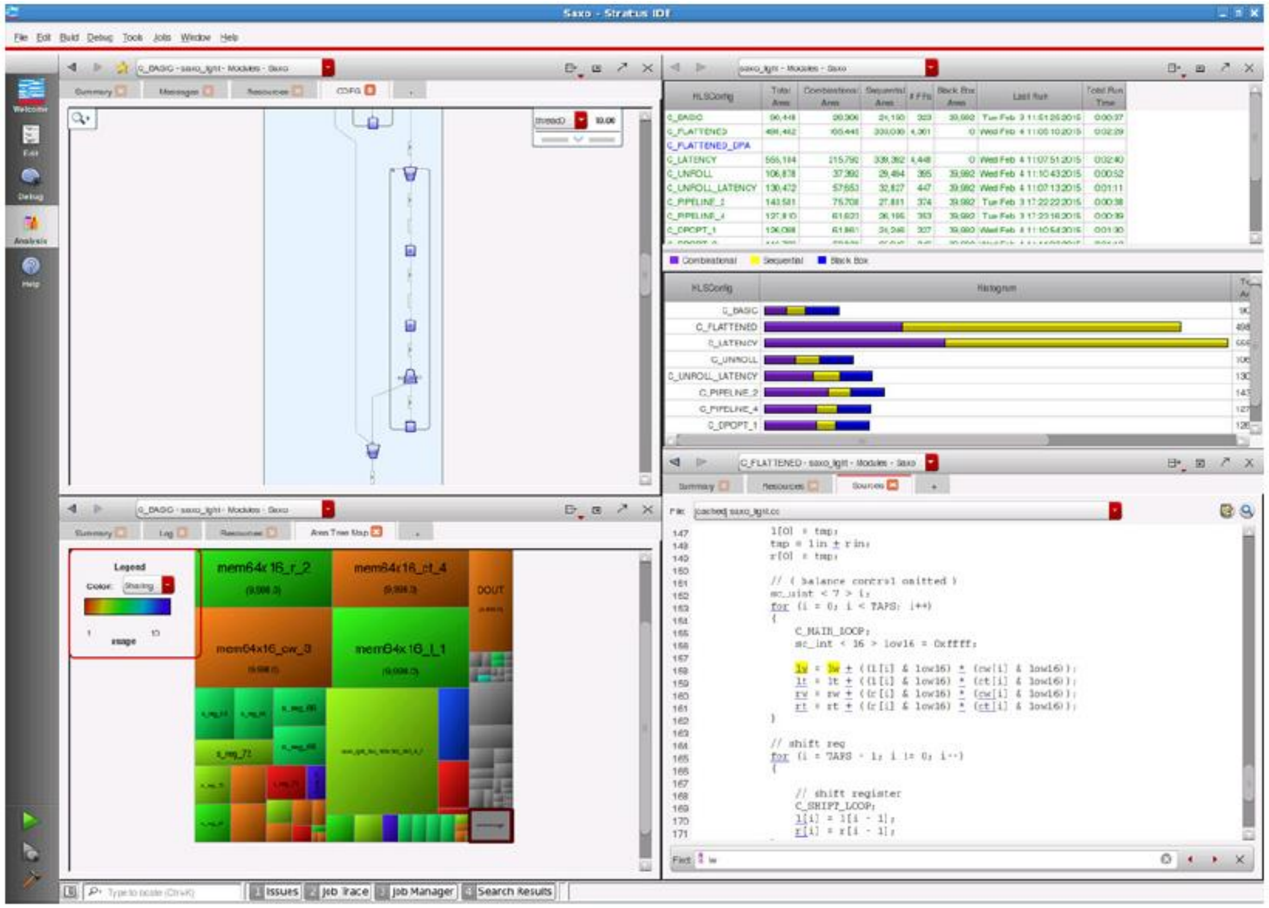
- Mentor Graphics→Calypt: Catapult C (Acquiqred by Calypto)→ Mentor Graphics Catapult C
- Forte Design System: Cynthesizer (Acquired by Cadence)
- Synopsys: Synphony C compiler
- Cadence: C2Silicon→Startus HLS
- ChipVision: PowerOpt?
- Xilinx: Vivado
- NEC CyberWorkBench

Cynthesizer → Startus HLS





Cynthesizer → Startus HLS



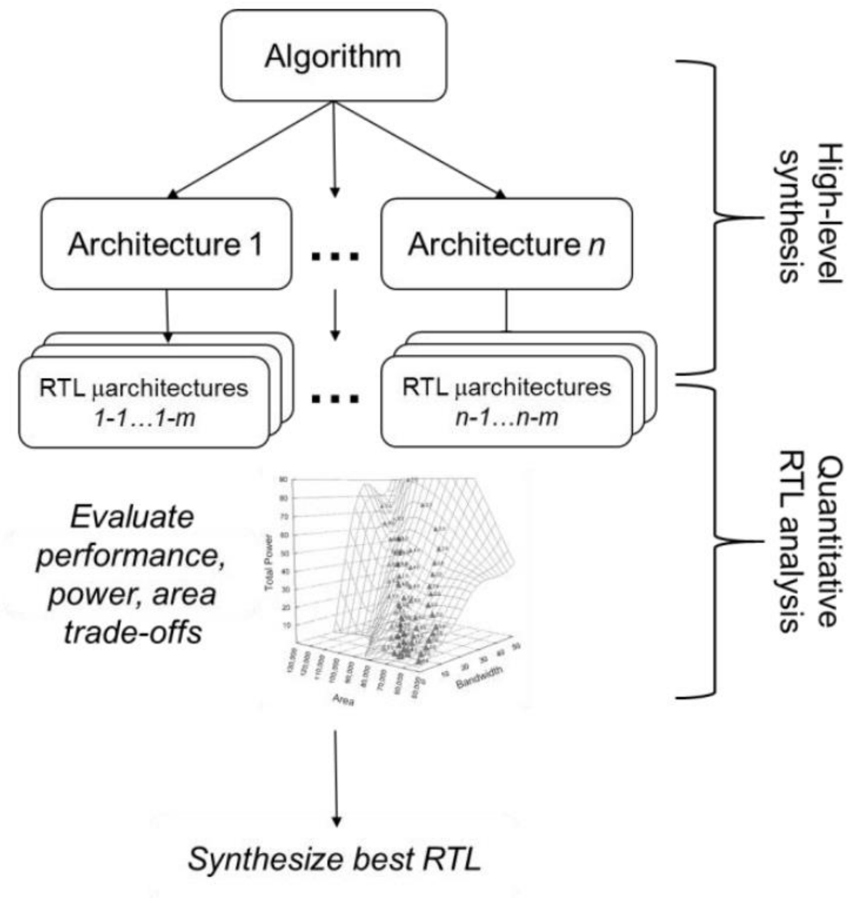
HLSConfig	Total Area	Combinational Area	Sequential Area	# FFs	Black Box Area	Last Run	Total Run Time
C_BASIC	96,448	20,306	24,150	323	39,992	Tue Feb 3 11:51:26 2015	0:00:37
C_FLATTENED	498,462	165,445	333,039	4,361	0	Wed Feb 4 11:05:10 2015	0:02:29
C_FLATTENED_DPA							
C_LATENCY	555,184	215,792	339,392	4,448	0	Wed Feb 4 11:07:51 2015	0:02:40
C_UNROLL	106,878	37,392	29,494	395	39,992	Wed Feb 4 11:10:43 2015	0:00:52
C_UNROLL_LATENCY	130,472	57,653	32,827	447	39,992	Wed Feb 4 11:07:13 2015	0:01:11
C_PIPELINE_2	143,581	75,708	27,881	374	39,992	Tue Feb 3 17:22:22 2015	0:00:38
C_PIPELINE_4	127,810	61,623	26,195	353	39,992	Tue Feb 3 17:23:16 2015	0:00:39
C_DPOPT_1	126,098	61,861	24,246	327	39,990	Wed Feb 4 11:10:54 2015	0:01:30

🟪 Combinational 🟨 Sequential 🟦 Black Box

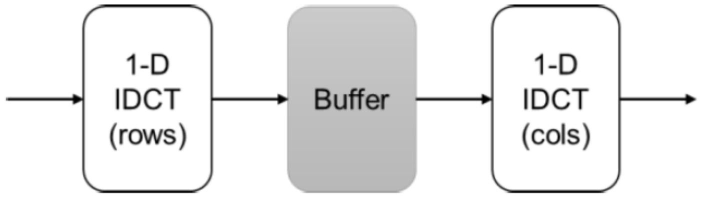
File: [cached] saxo_lgnt.cc

```
l[0] = tmp;
tmp = lin + rin;
r[0] = tmp;

// ( balance control omitted )
sc_uint < 7 > i;
for (i = 0; i < TAPS; i++)
{
    C_MAIN_LOOP;
    sc_int < 16 > low16 = 0xffff;

    lw = lw + ((l[i] & low16) * (cw[i] & low16));
    lt = lt + ((l[i] & low16) * (ct[i] & low16));
    rw = rw + ((r[i] & low16) * (cw[i] & low16));
    rt = rt + ((r[i] & low16) * (ct[i] & low16));
}

// shift reg
for (i = TAPS - 1; i != 0; i--)
{

    // shift register
    C_SHIFT_LOOP;
    l[i] = l[i - 1];
    r[i] = r[i - 1];
```

New Design Methodology with HLS

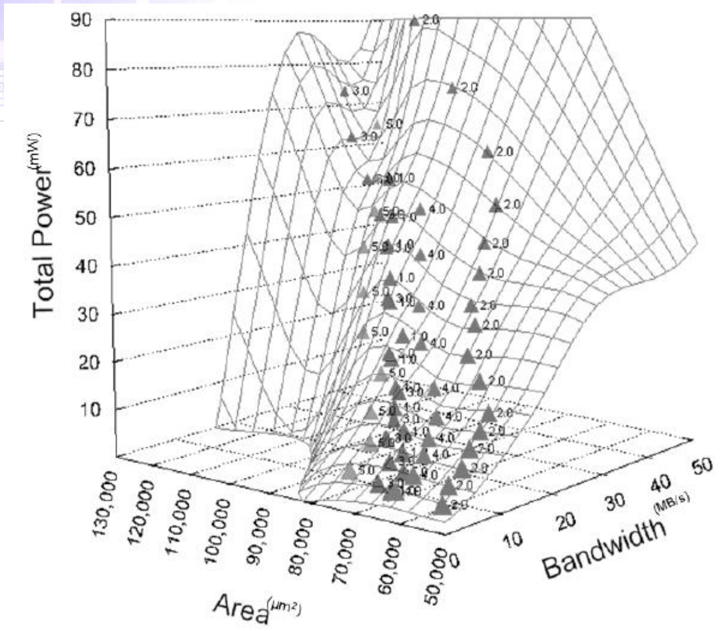


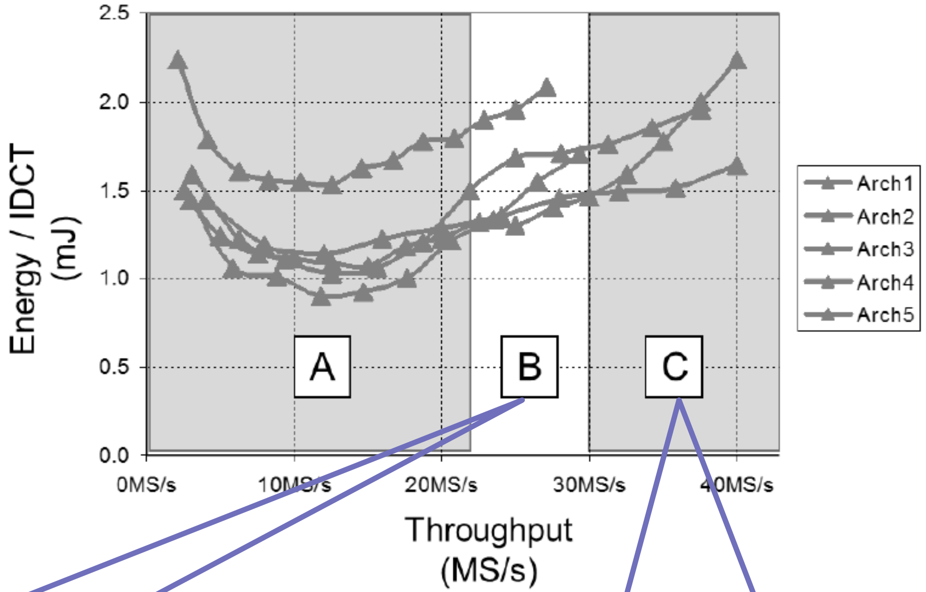
Algorithm
Architecture 1
...
Architecture n
RTL µarchitectures 1-1...1-m
...
RTL µarchitectures n-1...n-m
High-level synthesis
Evaluate performance, power, area trade-offs
Total Power
Area
Bandwidth
Quantitative RTL analysis
Synthesize best RTL



- Implement the 2-D DCT with 61 different micro-architectures
  - Buffer architecture
  - Latency
  - Loop pipelining
  - Clock frequency

New Design Methodology with HLS

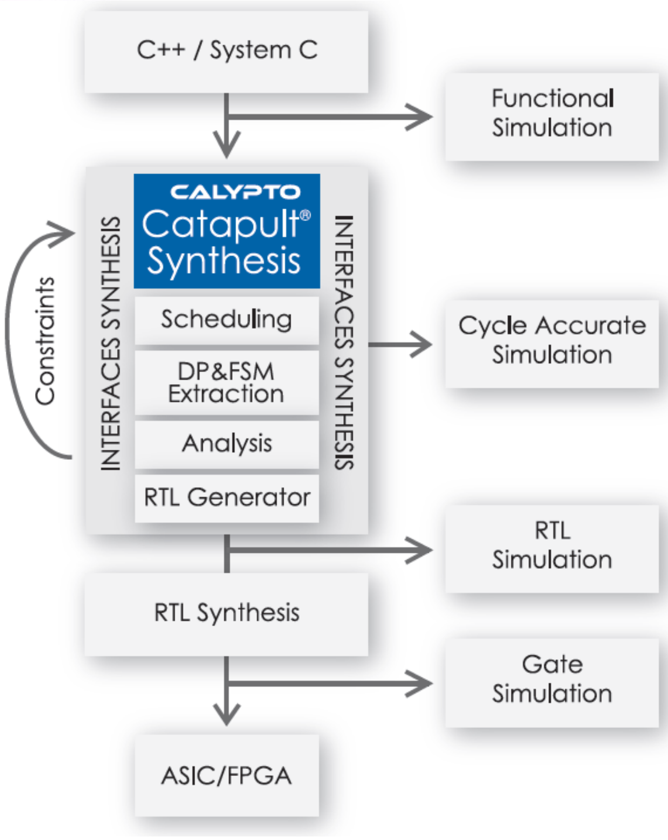


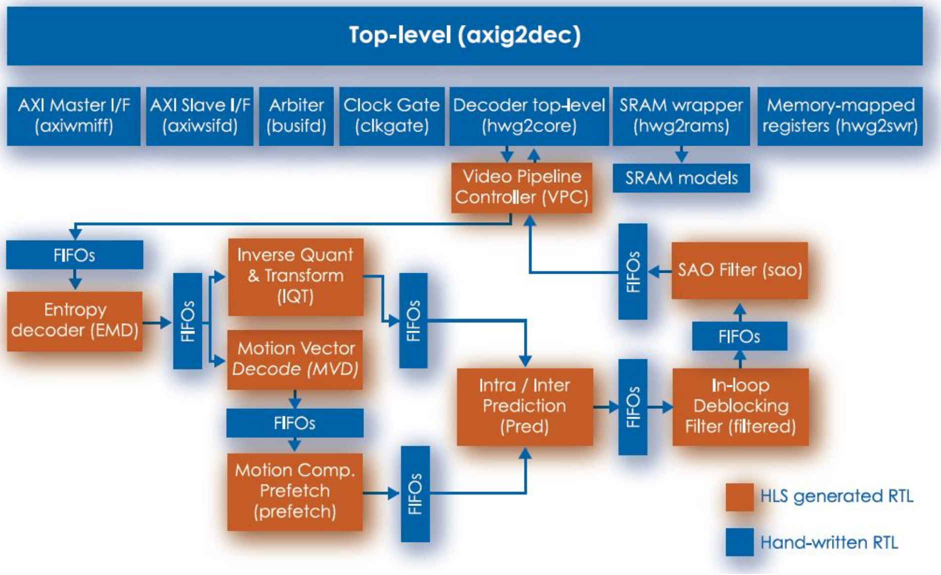


Unpipelined microarchitecture taking 16-clock cycles per loop iteration

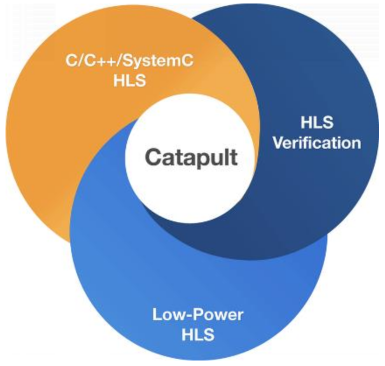
Highly pipelined microarchitecture

Menter Graphics Catapult

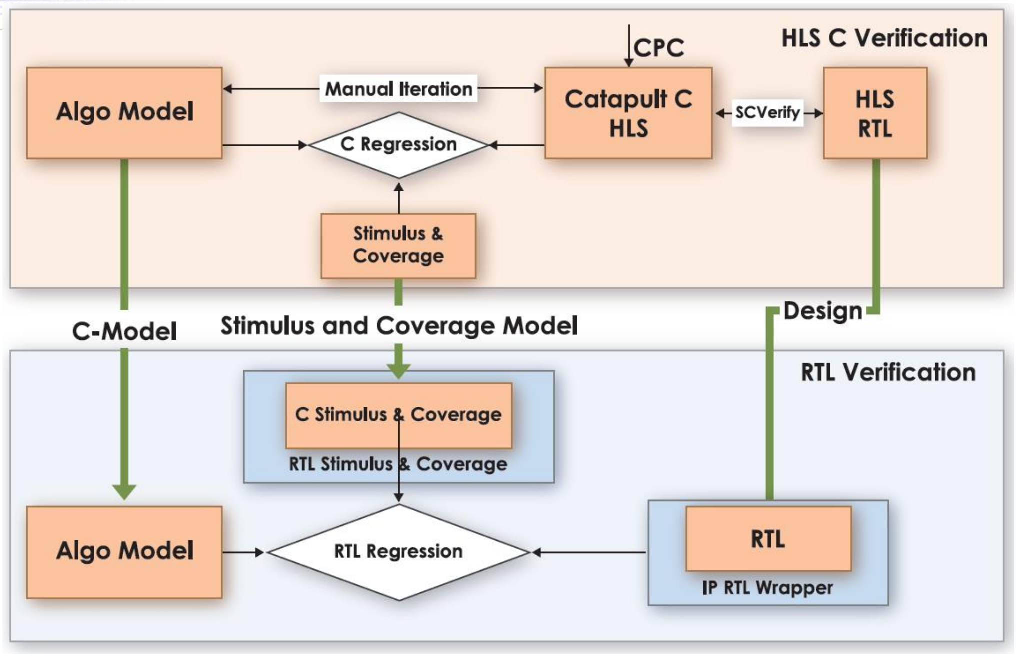



Top-level (axig2dec)
AXI Master I/F (axiwmiff)
AXI Slave I/F (axiwsifd)
Arbiter (busifd)
Clock Gate (clkgate)
Decoder top-level (hwg2core)
SRAM wrapper (hwg2rams)
Memory-mapped registers (hwg2swr)
Video Pipeline Controller (VPC)
SRAM models
FIFOs
Entropy decoder (EMD)
FIFOs
Inverse Quant & Transform (IQT)
Motion Vector Decode (MVD)
FIFOs
FIFOs
Motion Comp. Prefetch (prefetch)
FIFOs
FIFOs
Intra / Inter Prediction (Pred)
FIFOs
SAO Filter (sao)
FIFOs
FIFOs
In-loop Deblocking Filter (filtered)
HLS generated RTL
Hand-written RTL

Google WebM Project

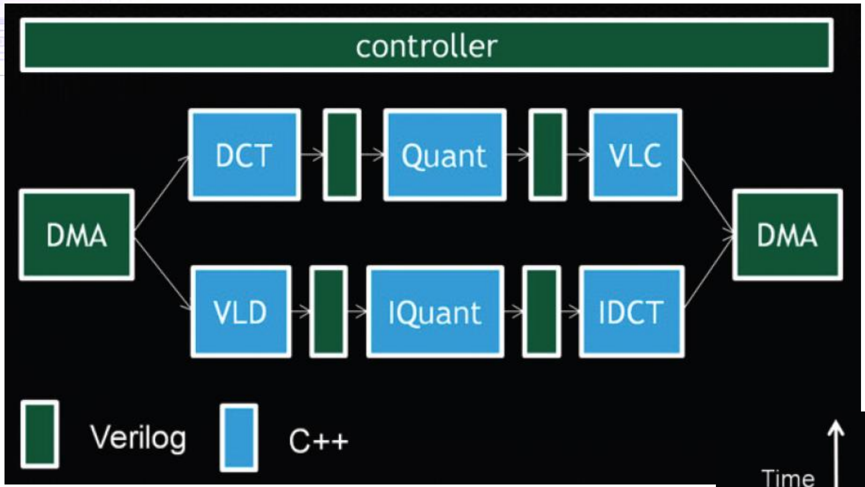


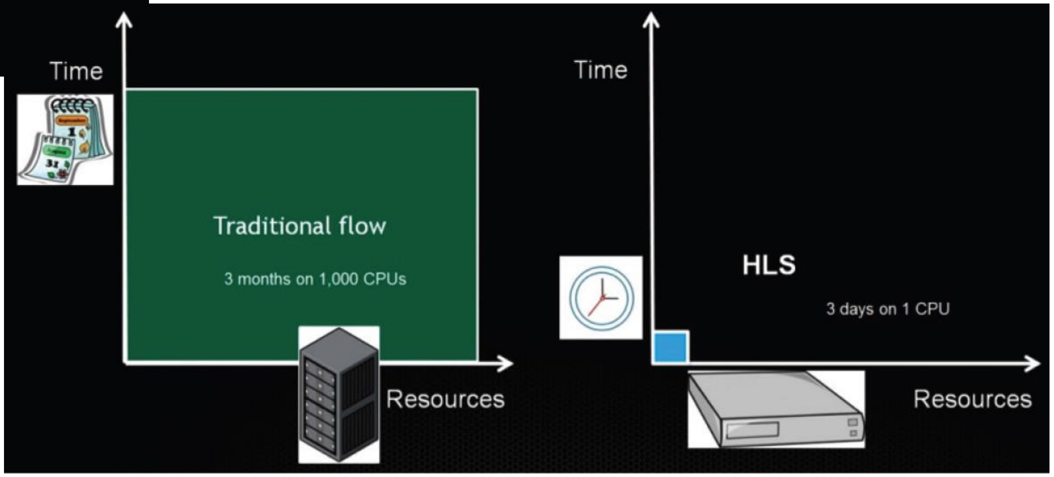
Example from Qualcomm



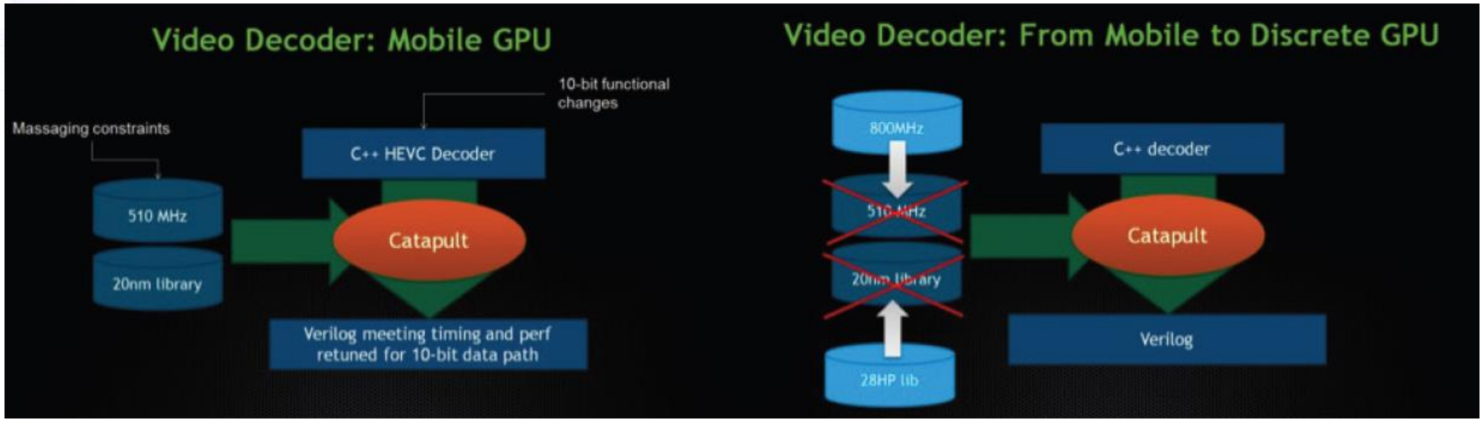
- **The HLS design code space is much smaller at the C-level** than at the RTL, making it easier to verify and correct; the **100x faster simulation speeds** enable us to detect problems and close coverage magnitudes faster than in RTL
- With the HLS methodology, what is verified in C stays verified in the RTL domain. As a result, most of the bugs are found and corrected in C.
- When HLS/HLV is done, the remaining work in the RTL environment is mostly at the interface level.

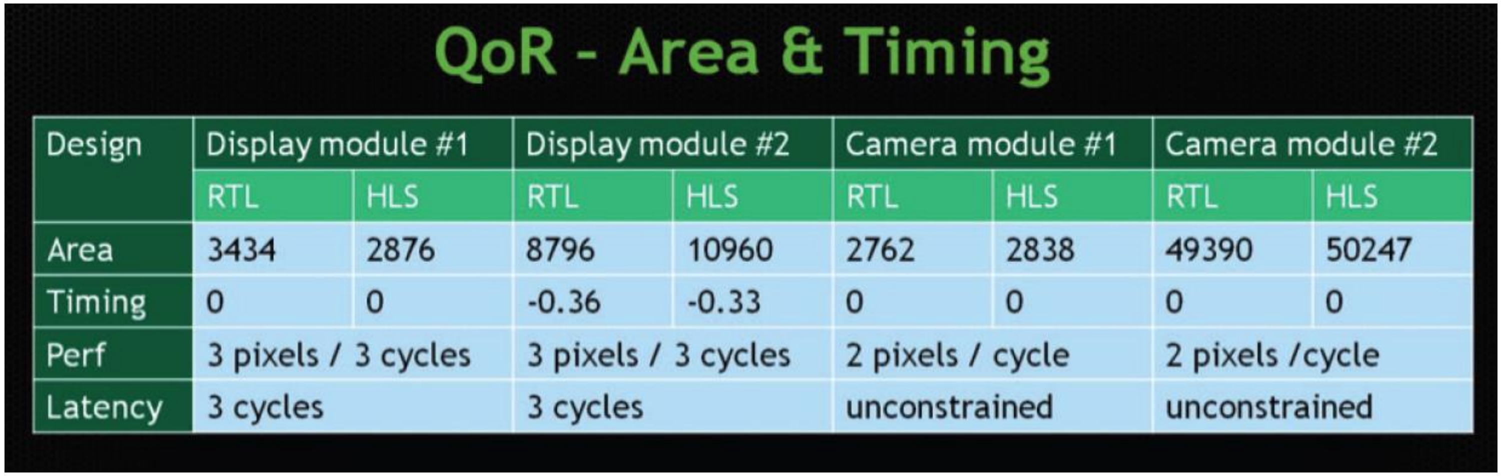
Example from NVIDIA (Image Decoder)





Example from NVIDIA (Image Decoder)


Video Decoder: Mobile GPU
10-bit functional changes
Massaging constraints
C++ HEVC Decoder
510 MHz
20nm library
Catapult
Verilog meeting timing and perf retuned for 10-bit data path
Video Decoder: From Mobile to Discrete GPU
800MHz
510 MHz
20nm library
28HP lib
C++ decoder
Catapult
Verilog



QoR - Area & Timing

Design	Display module #1		Display module #2		Camera module #1		Camera module #2	
	RTL	HLS	RTL	HLS	RTL	HLS	RTL	HLS
Area	3434	2876	8796	10960	2762	2838	49390	50247
Timing	0	0	-0.36	-0.33	0	0	0	0
Perf	3 pixels / 3 cycles		3 pixels / 3 cycles		2 pixels / cycle		2 pixels /cycle	
Latency	3 cycles		3 cycles		unconstrained		unconstrained	

Outline

- Introduction to SoC
- Relationship between SoC and multimedia systems
- Challenges for SoC Design
- SoC design methodologies
- New SoC design methodologies: ESL
- Modeling issues
- Some existing system-level design tools
- Conclusion

Conclusion (1)

- Multimedia systems will be one of the most important applications of SoC
- SoC can be designed efficiently with System-Level Design methodology
  - Can reduce iterations
  - Quick architecture closure
  - Hardware/Software co-design in early stage

Conclusion (2)

- Modeling is important in System-Level Design
  - Among different levels, Transaction-Level Modeling is the most important
  - Many languages can be used to develop the models
  - SystemC and SystemVerilog can be used for different levels

Conclusion (3)

- Many commercial ESL tools are available
  - Synopsys's solution
  - High level synthesis tools
  - In-house tools can also be developed

References

- 李昆忠、簡韶逸、鄺獻榮、邱瀝毅、郭致宏，電子系統層級設計教授，教育部顧問室「超大型積體電路與系統設計教育改進」計畫SLD聯盟。
- Y.-A. Chen, “Functional verification for system-on-chip (SoC) designs,” *Slides of System-Level Modeling for System-on-a-Chip Design Workshop.*
- R. Bergamaschi, “System-level design in practice,” *Slides of System-Level Modeling for System-on-a-Chip Design Workshop.*
- 董蘭榮, “晶片系統簡介,” 教育部晶片系統設計概論教材.
- M. Keating and P. Bricaud, *Reuse Methedology Manual*, 3rd Ed., Kluwer Academic Publishers, 2002.
- G. Martin and H. Chang Ed., *Wining the SoC Revolution*, Kluwer Academic Publishers, 2003.
- Documents from ARM, http://www.arm.com
- Documents from Synopsys, http://www.synopsys.com