

The Reality of System Design Today: Do Theory and Practice Meet?

Grant Martin

Fellow, Cadence Berkeley Labs

ACSD, Portugal, 18 June 2003: 0900-1000

ACS

Third International Conference

Point of view

- The history of system-level design as a viable commercial concern for the EDA tools industry is littered with false starts:
 - ESDA: Electronic System Design Automation
 - Behavioural synthesis
 - ESL: Electronic System-Level design
- In 2003, EDA is continuing to shrink its focus to primarily physically-related SoC design problems at 130-90-sub-90 nm processes.
- Yet many embedded systems today present profound problems of specification, design and verification:
 - Wireless and wired communications terminals and infrastructure
 - Multimedia consumer devices
 - Large complex systems

Point of view, continued

- Is the problem one of:
 - The system design community is too small and diverse, thus making a commercial marketplace for tools permanently unviable?
 - Have we espoused incorrect theories about how these systems should be designed and verified?
 - Have we been premature in trying to ‘industrialise’ system-level design?
 - Have we been looking in the wrong place for a large enough community of designers with compelling problems which can be solved on a commercial basis?
 - Or a combination of all of the above.....?
- We will examine a number of issues in this talk

Outline

- SoC and System Level Design
- Key Requirements
- An existential view of HW-SW Codesign
- Algorithmic Design and Implementation
- Modelling and Design of SoCs
- System Virtual Prototypes of SoCs for ESW
- What about Behavioural Synthesis?
- New SoC Architectures
- Conclusion

- “System” is more important than “Chip”
- Today’s chipset = tomorrow’s chip or SiP
- The system must be designed as an entity with tradeoffs across boundaries: HW-SW, analogue-digital, chip-package-board

System-Level Design:

- Always tomorrow’s methodology
- EDA’s focus is shrinking to IC Physical design
- SoC may be the best place to see system design applied

Concurrency:

- Most interesting embedded systems are fairly concurrent and becoming more so:
 - Multiple threads of control
 - Multiple dataflow processing streams
 - Multiple RISC + DSP ($1+1 \Rightarrow n+m$)
 - But designers are afraid of concurrency at the system level

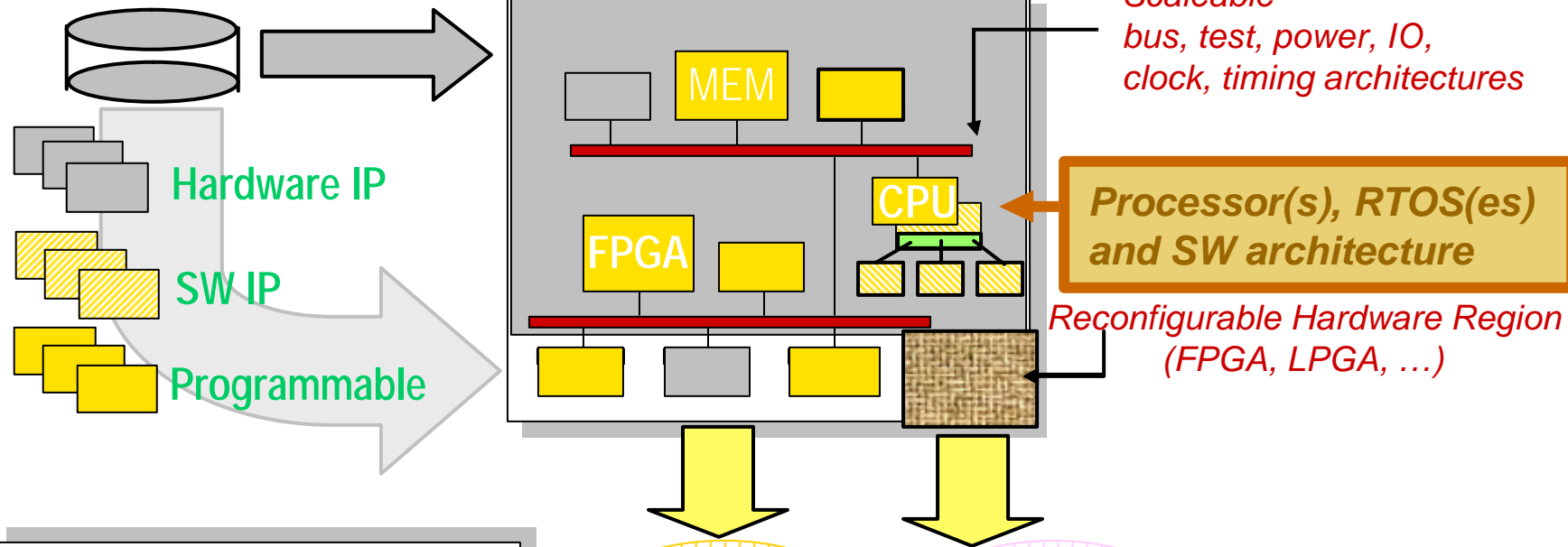
Key SoC/System Level Design Requirements

- Algorithmic design and implementation
- Modelling of SoCs and SoC platforms at the system level
 - Integration of SoC and configuration of designs
 - Build a platform model and verify HW-SW interfaces
 - Performance analysis and Design Space Exploration
- “System Virtual Prototypes” for embedded SW:
 - Hardware-dependent SW (HdS)
 - ESW application development

SoC Platforms

Pre-Qualified/Verified
Foundation-IP*

HW-SW Kernel + Reference Design



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

An existential view of HW-SW Codesign

- Does it exist?....i.e. as a delayed implementation choice
- Tradeoffs of HW vs. SW – not very relevant for most designs
 - Legacy: most tradeoffs are known or dead obvious
 - Processors are changed only very rarely
 - The SW legacy is enormous
 - Specifications easily drive an obvious choice of HW implementation when needed
- More relevant?
 - “SW-SW Co-Design” – mapping functions to multiple programmable or configurable computation and communications resources
 - Obvious need for concurrency-based design methods and tools!

Algorithmic Design and Implementation

- Design and Implementation of complex control and dataflow algorithms in HW, SW or a combination
- Today's best practices use system level design tools
- Well established for many years
 - Dataflow is better handled than mixes of dataflow and control
- Used both for less integrated systems as well as SoC

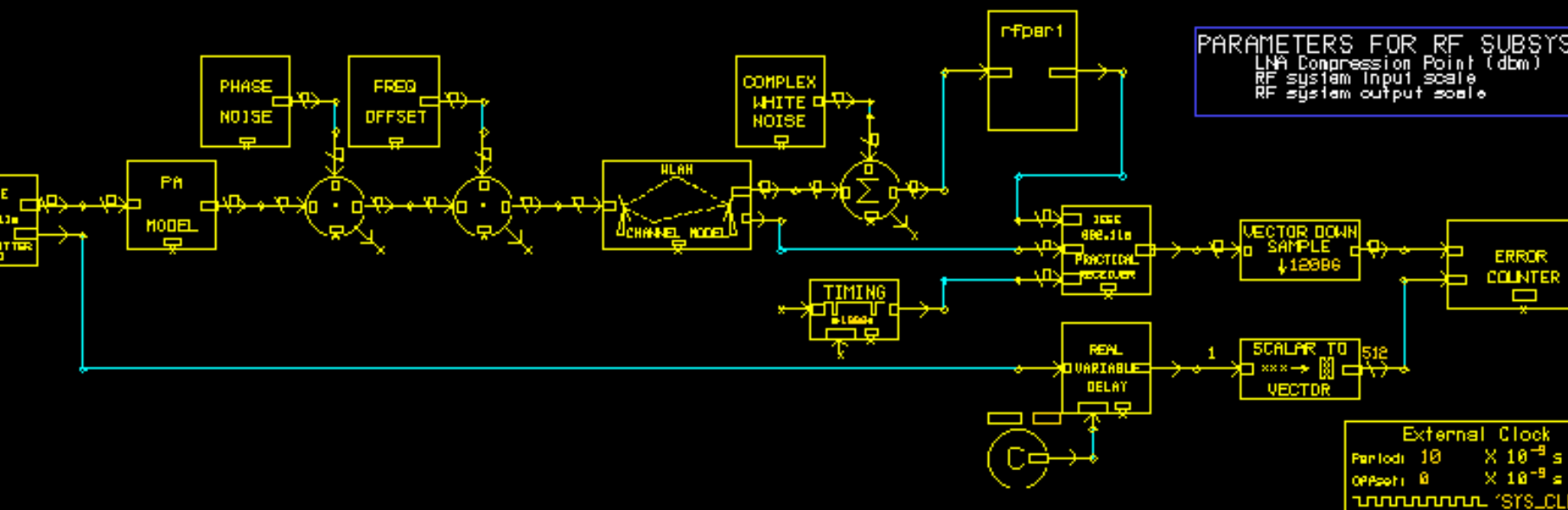
Dataflow algorithms

- Classic and well-established tools exist
 - Mathworks: Matlab, Simulink
 - Cadence: SPW
 - Research: Ptolemy I/II
 - Synopsys: COSSAP/CoCentric System Studio

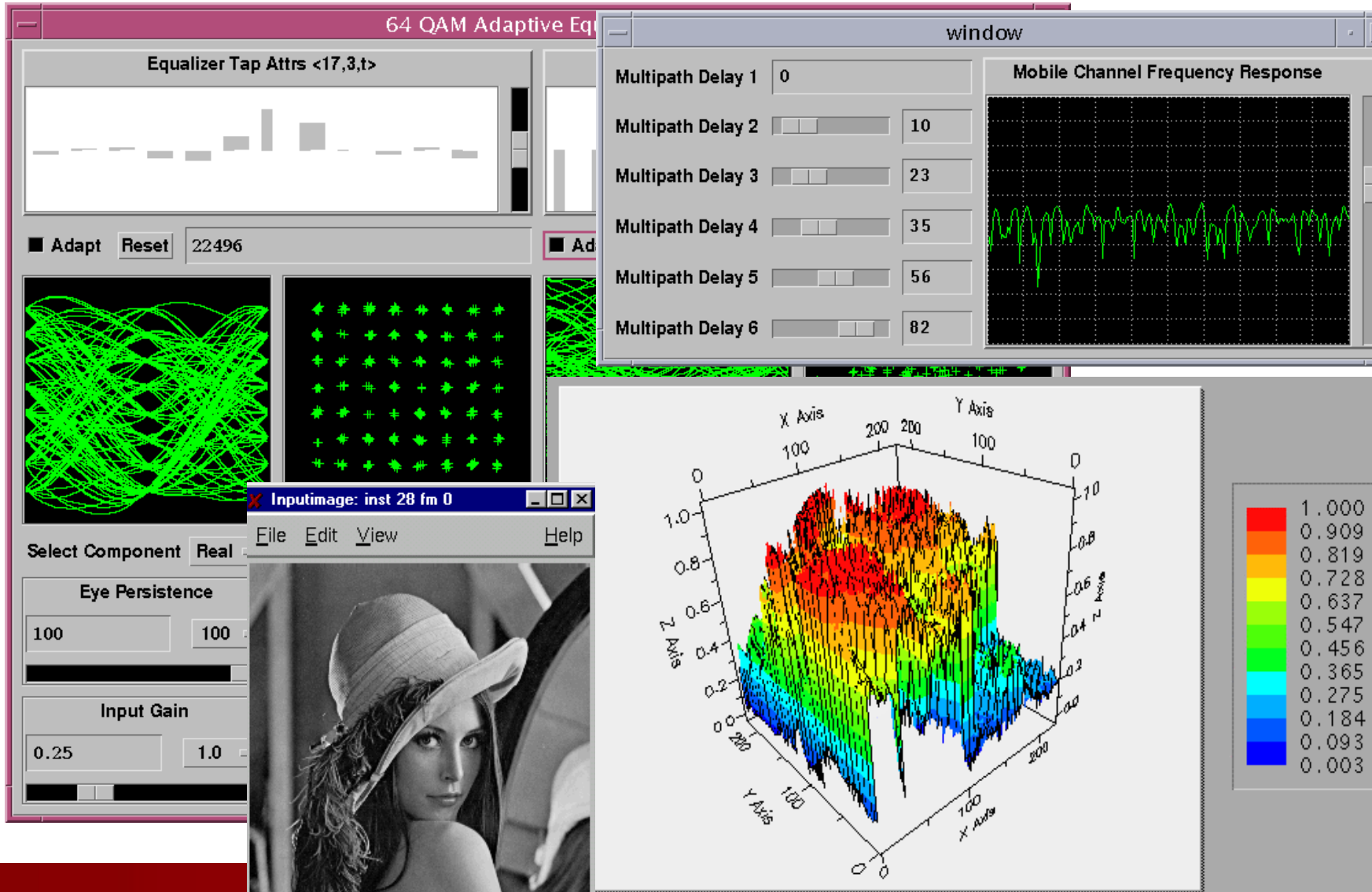
A Dataflow tool example

E 802.11a ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING (OFDM) SYSTEM IN 5 GHz BAND WITH IDEAL AND PRACTICAL RECEIVERS

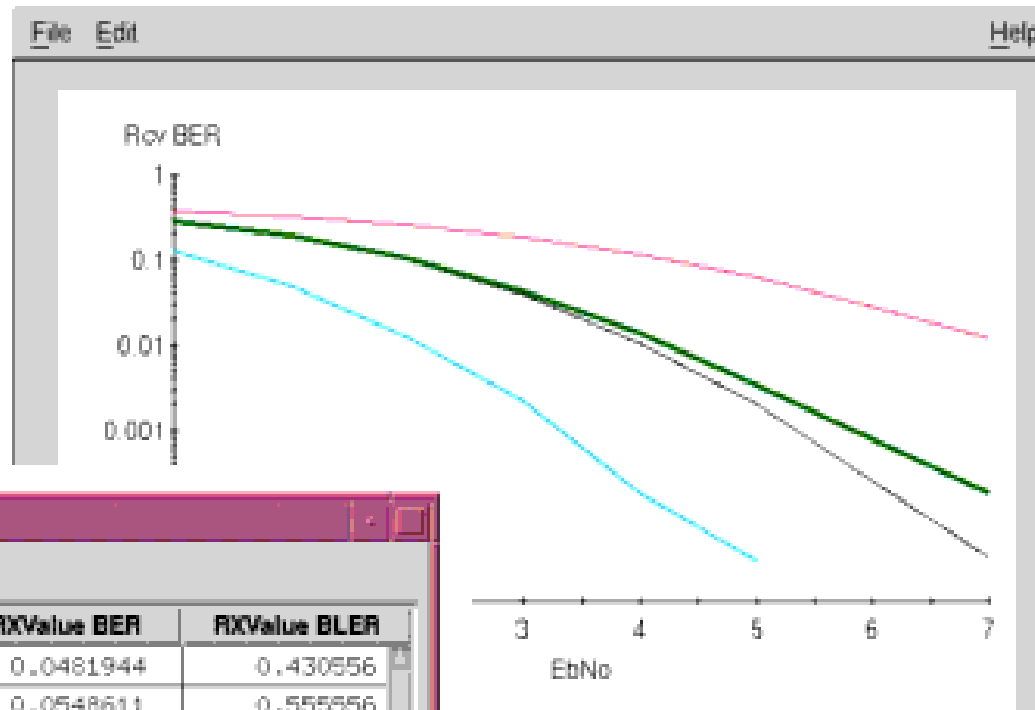
FIXED PARAMETERS:		UNEDITABLE PARAMETERS:		PRACTICAL RECEIVER PARAMETERS:	
Data Rate (Mbps)	12	SERVICE bits	16	Type of receiver	Practical
PSDU length	64	TAIL bits	6	Channel estimation	Practical
Eb/N0 (dB)	11.0	PAD bits	42	Frequency offset (KHz)	100.0
Decoding	soft	Data bits	576	Frequency offset compensation	Ideal
Channel	Fading	Modulation	QPSK	RMS phase noise (deg)	1.0
Delay spread (ns)	50.0	Coding rate	1/2	Phase noise bandwidth (3dB)	10e3
Number of paths	8	N_BPSC	2	Carrier phase correction	On
Fading over packets	fix	N_CBPS	96	PA model	Ideal
Idle symbols (>1)	2	N_DBPS	48	Stop condition	Errors (p)
OTHER PARAMETERS (EDITABLE):		Number of OFDM symbols	12	Number of errors to stop	10
Time windowing	no	SNR (dB)	8.781		
Example mode	no				



A Dataflow Tool: Interactive Simulation



Bit Error Rate Analysis



Multisimulation Probe Results

ber_sim/bler.csv

EbNo	Metric	phase	RXValue Blocks	RXValue BER	RXValue BLER
3	Hard	3	72	0.0481944	0.430556
1	Soft	3	72	0.0548611	0.555556
2	Soft	3	72	0.0122222	0.208333
1	Hard	3	72	0.164722	0.944444
3	Soft	3	72	0.000277778	0.0138889
4	Hard	3	72	0.00875	0.194444

Plot Type: Y = F(X)

X: RXValue BER

Y: RXValue BLER

New Trace for Each: []

Plot Points Where: []

Plot

Metric=Hard, phase=2
 Metric=Hard, phase=3
 Metric=Soft, phase=2
 Metric=Soft, phase=3

lay = # of Pts =
 Freq = Point =
 Hue = Y Value =

Control algorithm capture and analysis

- State Machine capture and simulation
 - Links to HW and SW generation
- A possible place for adapting SW development flows into the system space
 - SDL, UML
 - Statecharts, state diagrams, message sequence charts
 - Esterel
 - Synchronous Reactive systems

Modelling and Design of SoCs and their architectures

- Capturing SoC architecture and providing SoC configurators
- SoC model integration and developing verification models
 - Especially HW/SW integration models
 - “Golden models” for implementation verification
- Design Space Exploration

SOPC Configurators – Example: Altera SOPC Builder

Altera SOPC Builder

File System Module View Help

System Contents More "nios_cpu" Settings System Generation

System Clock Frequency: 80.00 MHz

Components

- ARM-based Excalibur CPU
- Altera Nios 2.0 CPU
- Interface to User Logic
- Bridges
 - Avalon Tri-State Bridge
 - PLD Applications Nios-P
 - AHB To Avalon Bridge
- Communication
 - SPI (3 Wire Serial)
 - UART (RS-232 serial po
 - M16550S Enhanced UAI
 - CAN 2.0 Network Contro
 - AHB Ethernet MAC
 - Altera AHB UART
- Memory
 - On-Chip Memory (RAM c
 - SDRAM Controller
 - SSRAM (Micron MT58L2
 - Flash Memory
 - SRAM (one or two IDT7
- Other
 - DMA
 - PIO (Parallel I/O)

Diagram labels:

- nios_cpu / instruction_master (avalon)
- nios_cpu / data_master (avalon)
- arm_922t_stripe (AHB)
- ahb_ethernet_mac_0 (AHB)
- tri_state_bridge_0 (avalon_tristate)

Use	Module Name	Description	Bus Type	Base	End	IRQ
<input checked="" type="checkbox"/>	arm_922t_stripe	ARM-based Excali...				
<input checked="" type="checkbox"/>	ahb_ethernet_...	AHB Ethernet MAC	AHB	0x00805000	0x00805FFF	19
<input checked="" type="checkbox"/>	nios_cpu	Altera Nios 2.0 CPU	avalon			
<input checked="" type="checkbox"/>	boot_monitor	On-Chip Memory (...)	avalon	0x00804400	0x008047FF	
<input checked="" type="checkbox"/>	uart1	UART (RS-232 seri...	avalon	0x00804000	0x0080401F	16
<input checked="" type="checkbox"/>	uart2_debug	UART (RS-232 seri...	avalon	0x00804020	0x0080403F	17
<input checked="" type="checkbox"/>	tri_state_bridg...	Avalon Tri-State Br...	avalon avalon_...			
<input checked="" type="checkbox"/>	ext_flash	Flash Memory	avalon_tristate	0x00808000	0x0080BFFF	
<input checked="" type="checkbox"/>	ext_ram	SRAM (one or two ...)	avalon_tristate	0x00840000	0x0087FFFF	
<input checked="" type="checkbox"/>	timer1	Interval timer	avalon	0x00804040	0x0080405F	18
<input checked="" type="checkbox"/>	pio_0	PIO (Parallel I/O)	avalon	0x00804060	0x0080406F	
<input checked="" type="checkbox"/>	pio_1	PIO (Parallel I/O)	avalon	0x00804070	0x0080407F	

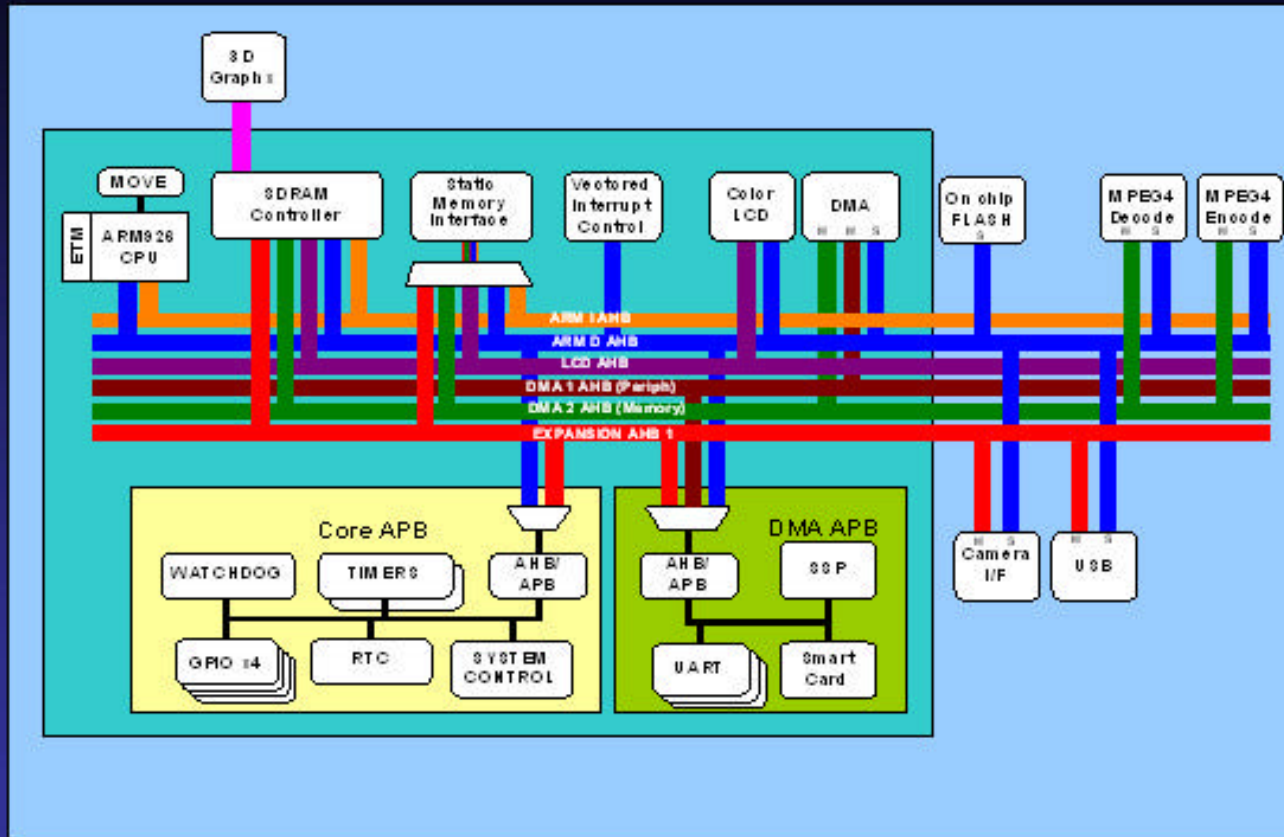
Buttons: Add... Move Up Move Down

Buttons: Exit < Prev Next > Generate

Source:
Altera website
www.altera.com

SoC Model Integration: SystemC

Platform design problem



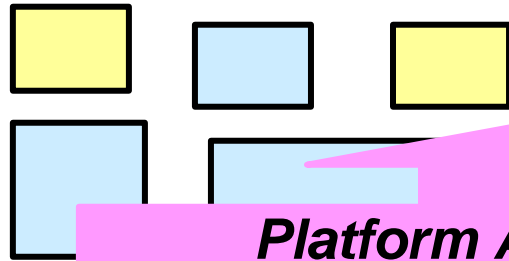
System Virtual Prototypes of SoCs for ESW

- Hardware-dependent SW developers
 - Need cycle and phase accuracy, bit-level precision
- Applications SW developers
 - Need functional correctness and fast execution

A Scenario for Embedded Systems



**Applications SW
Designers**

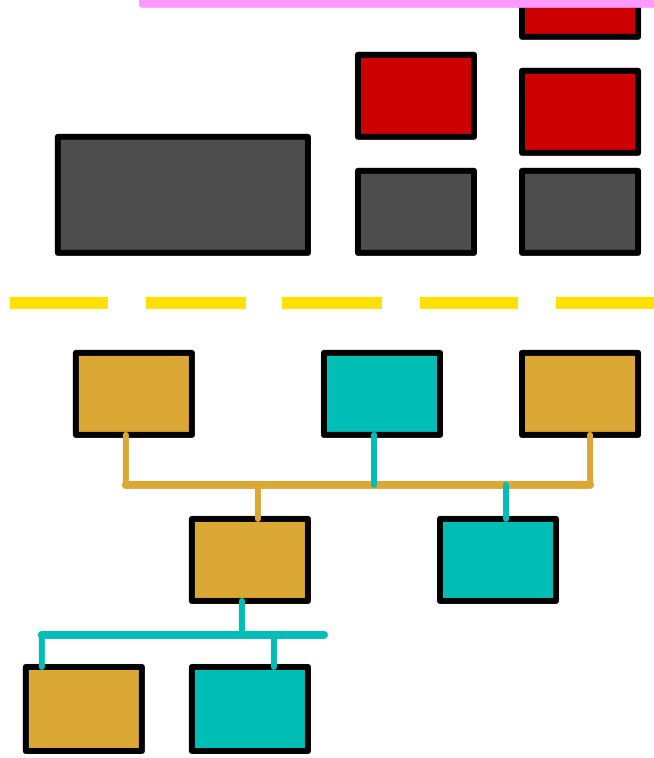


**Platform Abstraction:
Link between the communities**

Platform Abstraction



**Systems
Architects**



**HdS SW
Designers**



HdS API



**HW
Designers**



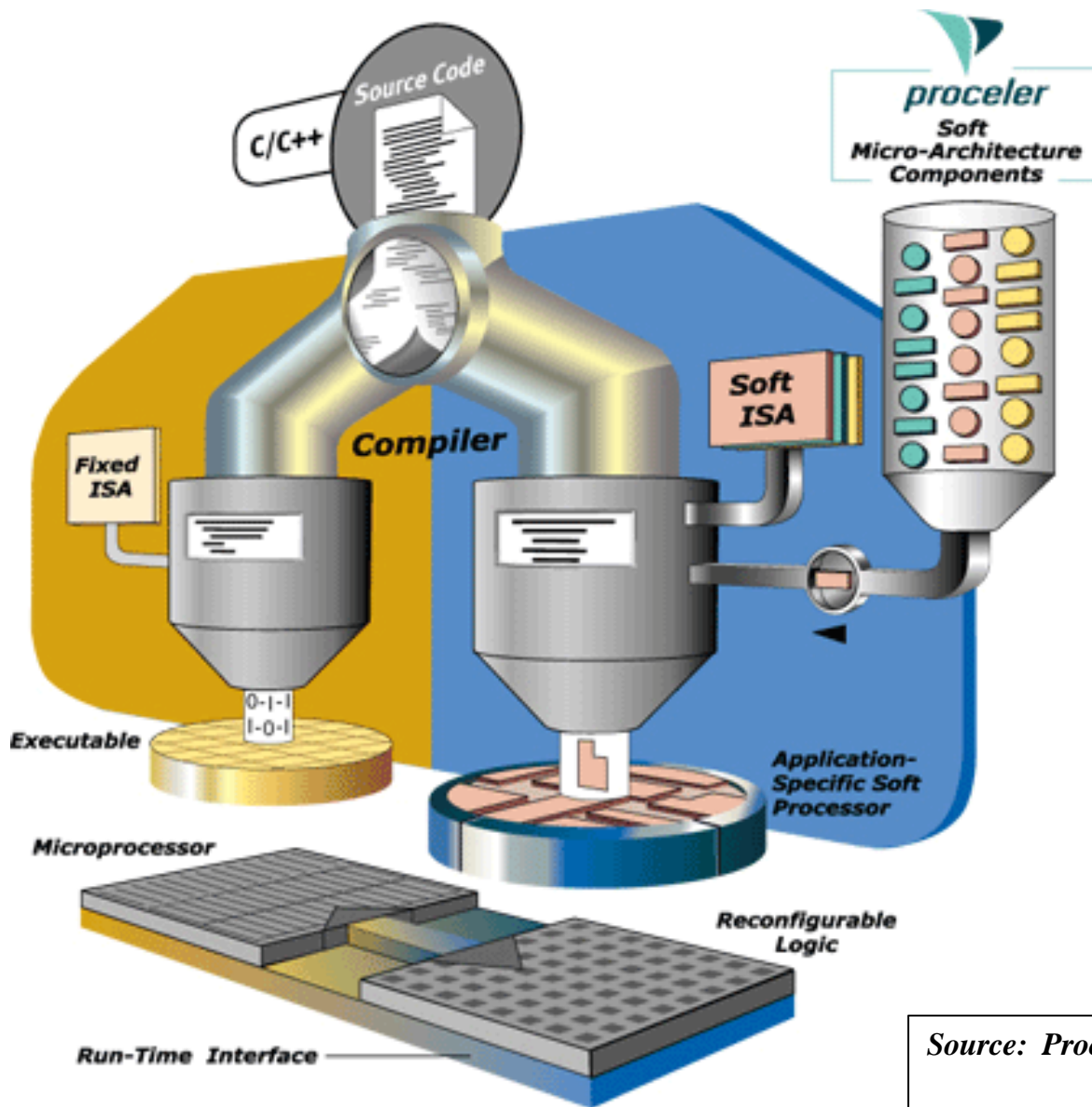
System Level Virtual Prototypes

- Develop HdS Models of your HW-SW platform
 - Using, for example, SystemC
 - SystemC 2.0 adds abstract communications modelling
 - SystemC 3.0 will add abstract RTOS modeling
 - Scheduling, communications services
- Use SystemC (2.0-3.0) as a simulation and analysis “Backbone” for analysing the complete HW-SW embedded system
 - An interoperable, Integration, Infrastructure
- Back-annotate the results INTO the SW developer’s development environment via fast-execution, functional System Virtual Prototypes
 - SW modeling tools
 - UML, SDL, ...
 - SW Integrated Development Environments

What about Behavioural Synthesis?

- Not been a major success for commercial tools
 - Quality of results uncompetitive with general RTL synthesis
- Used in specialised contexts
 - E.g. dataflow algorithm to implementation
- May resurface via ***SW implementation optimisation*** to processor+HW combination
 - Emerging “co-processor synthesis”

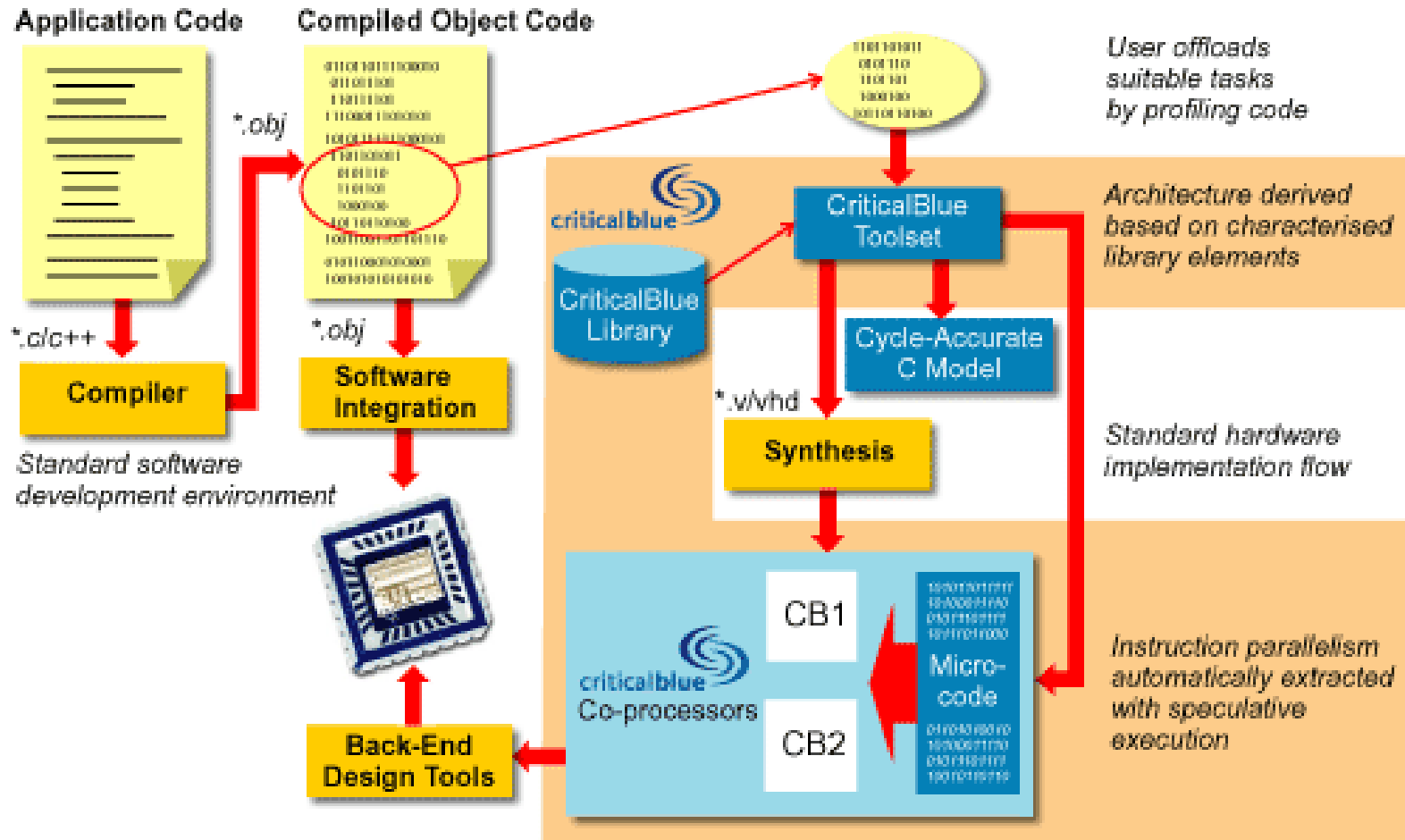
Compile to SW+HW: The Proceler Example



Source: Proceler Web Site www.proceler.com

(Now Unfortunately Defunct)

From the defunct to the emergent: Co-processor Synthesis - CriticalBlue



New SoC Architectures

- A 'sea' of Flexible, configurable computational resources
- Using flexible, configurable, on-chip communications networks
- New architectures require new thinking
- ***This*** may be the real opportunity for system-level design
- But it may come with a real SW focus
- Current interest in compiling SW models to processor+accelerating HW (often using reconfigurable logic)
- Obvious need here for Methods and Tools involving concurrency!
 - Mapping computation to sea of resources
 - Optimising configuration
 - Mapping communications to network
 - Making sure it all works together harmoniously

Software Washing Machine [Catthoor, IMEC]

Target independent source to source transformations
for data-intensive applications save power!

IN: Dirty C++ software IP's + scenario

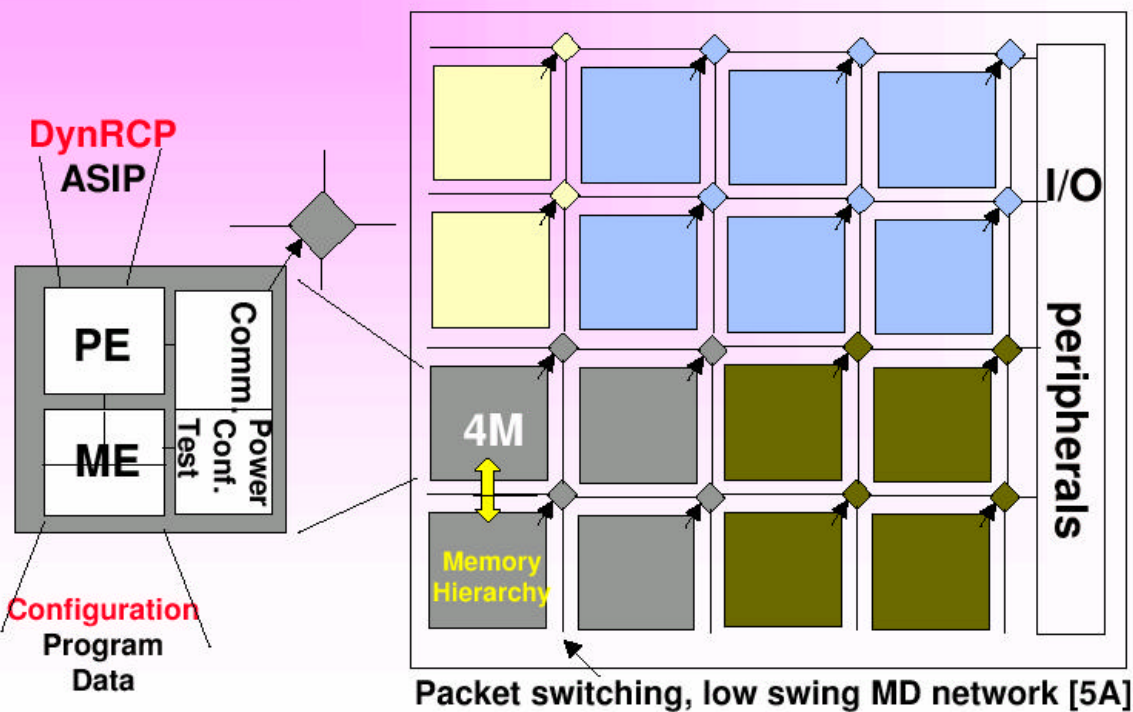


OUT: Cleaned Concurrent C++ tasks OUT (SYSTEMC)

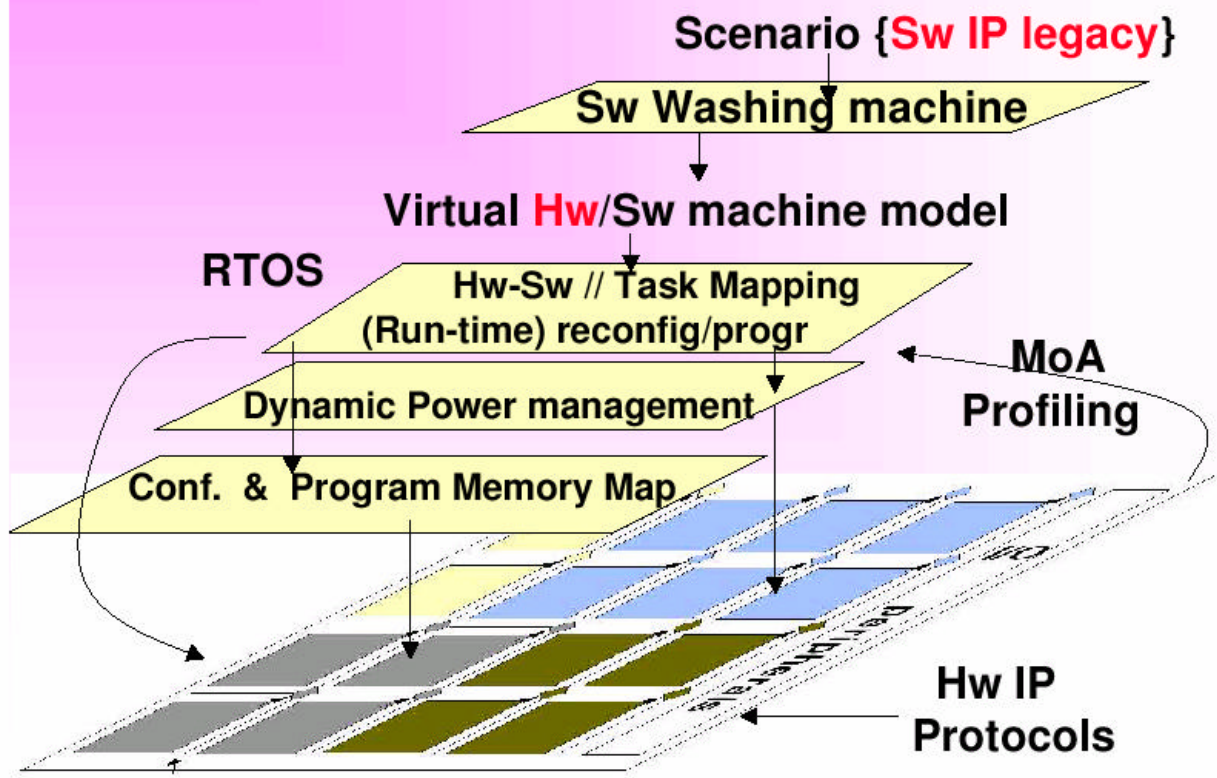
Platform specific compiler

Networked Adaptive Computing Machines

Subnetworks for compute intense concurrent tasks (brain zones)



Challenge#2: The devil is in the software



Conclusion

- Is this just a SW problem?
- Will systems design = SW design and implementation?
- Why should a system designer care about HW at all?
 - Except inasmuch that it gives him or her **choices** about implementation **tradeoffs**
 - ***This is a job for a compiler with optimisation options***
- What are the implications for the research community?
 - Can advanced tools and methods overcome designers fear of concurrency?
 - Can we also unify the HW and SW design community into a new one: **system designers** who, armed with the right tools, boldly, reliably and quickly can implement highly concurrent applications on networks of configurable processors?